# Structural Methods for the Synthesis of Speed-Independent Circuits

Enric Pastor, Jordi Cortadella, *Member, IEEE*, Alex Kondratyev, *Member, IEEE*, and Oriol Roig

***Abstract*—Asynchronous circuits can be modeled as concurrent systems in which events are interpreted as signal transitions. The synthesis of concurrent systems implies the analysis of a vast state space that often requires computationally expensive methods. This work presents new methods for the synthesis of speed-independent circuits from a new perspective, overcoming both the analysis and computation complexity bottlenecks.**

**The circuits are specified by *free-choice signal transition graphs* (STG's), a subclass of interpreted Petri nets. The synthesis approach is divided into the following steps: correctness, binary coding, implementability conditions, and logic synthesis. Each step is efficiently implemented by applying a set of structural techniques that analyze STG's without explicitly enumerating the underlying state space.**

**Experimental results show that circuits can be generated from specifications that exceed in several orders of magnitude the largest STG's ever synthesized—with over $10^{27}$ states. Computation times are also dramatically reduced. Nevertheless, the quality of results does not suffer from the use of structural techniques.**

***Index Terms*— Asynchronous circuits, Petri nets, speed-independent synthesis.**

## I. Introduction

ASYNCHRONOUS circuits promise a number of important advantages for the design of large digital circuits. Their modularity, potential low-power consumption, average-case computation time, and elimination of the clock distribution problem have encouraged their extensive analysis. However, any asynchronous implementation must satisfy much more restrictive conditions than its synchronous counterpart. Asynchronous circuits must be not only functionally equivalent to the specification but also free of *hazards*—undesired switching activity due to the skew of gate delays.

*Speed-independent circuits* (SI circuits) is a broadly used design style for asynchronous implementations. SI circuits rely on the *unbounded gate delay* model, which assumes unknown but finite delays on the gates, and *skew* at the wires bounded by the delay of the fastest gate. Thus, the correctness of the circuit requires the assumption that some wire forks are *isochronic* [1]. SI circuits are robust to parameter variations, i.e., the response time of an SI circuit subjected to temperature or voltage modifications may vary, but the circuit keeps working correctly. Additionally, an SI circuit does not need any modification to guarantee its correctness after a technology migration (the validity of isochronic forks must be checked, however). The most robust delay model, *delay-insensitive* circuits, also assumes unbounded wire delays. Unfortunately, the class of delay-insensitive circuits is very small from the practical point of view [1].

A wide range of synthesis techniques for asynchronous circuits rely on *event-based* models, such as Petri nets (PN's) [2] or *change diagrams* [3]. PN's are a powerful formalism to model concurrent systems that gracefully captures the notions of causality, concurrency, and conflict between events. As a model, their most interesting feature is the capability of implicitly describing a vast state space by a succinct representation. Hence, PN's have been chosen by many authors as a formalism to describe the behavior of asynchronous circuits by interpreting the events as signal transitions, thus coining the term *signal transition graph* (STG) [4], [5].

Each reachable marking of an STG has assigned a binary vector with the value of the circuit signals in that marking. Deriving logic equations from an STG requires the generation of the binary codes for all markings. Currently, most synthesis tools [6]–[8] perform an exhaustive token flow analysis to obtain the complete reachability graph of the PN and all binary vectors. Unfortunately, the reachability graph of highly concurrent systems can be exponential in the size of the STG that leads to the well-known *state explosion* problem. Some efforts have been devoted to propose structural methods for synthesis [9], [10], but they have been usually devised for restricted classes of PN's that compromise the potential expressiveness of this formalism.

This work presents a structural methodology for the synthesis of SI circuits from STG's. The proposed techniques have polynomial complexity if the underlying PN is free choice [11], [12], and can be efficiently extended to the class of PN's that can be covered by state machines [13].

The proposed structural techniques are based on the analysis of the *concurrency* relations of STG's [5], and the generation of covering cubes that approximate the reachable markings. Additional information obtained from the state machines of the STG allows one to refine the initial covering cubes, increasing the accuracy of the approximations. This methodology eliminates the *state explosion* problem by avoiding the explicit generation of all the markings in the STG. Even though

Manuscript received November 13, 1997; revised March 30, 1998. This work was supported in part by CICYT under Grant TIC98-0410. This paper was recommended by Associate Editor A. Saldanha.

E. Pastor is with the Department of Computer Architecture, Universitat Politécnica de Catalunya, Barcelona 08034 Spain (e-mail: enric@ac.upc.es).

J. Cortadella is with the Department of Software, Universitat Politécnica de Catalunya, Barcelona 08034 Spain (e-mail: jordic@lsi.upc.es).

A. Kondratyev is with the Computer Architecture Laboratory, University of Aizu, Aizu-Wakamatsu 965 Japan (e-mail: kondraty@u-aizu.ac.jp).

O. Roig is with National Semiconductor Corp., Santa Clara, CA 95052 USA (e-mail: Oriol.Roig@nsc.com).

Publisher Item Identifier S 0278-0070(98)08591-1.

the concurrency relations have been previously applied for synthesis [10], [14], this work generalizes the use of these relations, reducing the gap between structural and state-based approaches.

We aim at complementing the existing tools by providing alternative and efficient synthesis algorithms for state-machine-coverable STG's, which account for a large number of STG's used for circuit design. The area and delay results of the SI circuits synthesized by applying our method are presented and compared with those obtained by previous synthesis tools.

This paper is organized as follows. The formal notions on Petri nets and signal transition graphs are presented in Section II. The implementability conditions of *speed-independent* circuits are analyzed in Section III. Section IV illustrates the structural synthesis framework and its efficiency by means of two examples. To avoid the *state explosion problem*, Section V proposes a method to derive approximations of the reachability graph from the structure of the STG. Section VI describes how Boolean functions can be obtained from these approximations. A strategy to increase the accuracy of such approximations is introduced in Section VII. The overall logic-minimization framework is described in Section VIII, and further minimizations are outlined in the Appendix. Several experimental results and efficiency analysis are presented in Section IX. Section X concludes this paper.

## II. Basic Notions and Definitions

In this section, we briefly recall some of the basic definitions on logic functions, Petri nets, and signal transition graphs. For more detailed information on these topics, we refer the reader to [5], [12], and [15]–[17].

### A. Logic Functions

An incompletely specified $n$-variable *logic function* is a mapping $f$: $\{0, 1\}^n \rightarrow \{0, 1, -\}$. Each element $\{0, 1\}^n$ is called a *vertex*. The set of vertices where $f$ evaluates to 1, 0, and $-$ are called on-, off-, and dc-sets and are denoted by on($f$), off($f$), and dc($f$), respectively. A *literal* is either a variable $x_i$ or its complement $\overline{x}_i$. A *cube* $c$ is a set of literals such that if $x_i \in c$, then $\overline{x}_i \notin c$, and vice versa. Cubes can also be represented as an element $\{0, 1, -\}^n$, in which value "0" denotes a complemented variable $\overline{x}_i$, value "1" denotes a variable $x_i$, and $-$ indicates that the variable is not in the cube. A *cover* is a set of implicants that contains the on-set and does not intersect with the off-set.

### B. Petri Nets and STG's

A PN is a four-tuple $\Sigma = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, M_o \rangle$, where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the flow relation, and $M_o$ is the initial marking. Given a node $u \in \mathcal{P} \cup \mathcal{T}$, its postset and preset are denoted by $u^\bullet$ and ${}^\bullet u$, respectively. A *marking* of a PN is an assignment of a nonnegative integer to each place. If $k$ is assigned to place $p$ by marking $M$, we will say that $p$ is marked with $k$ tokens, i.e., $M(p) = k$. A *path* in a PN is a sequence $u_1 \cdots u_r$ of nodes such that $\forall\, i, 1 \leq i < r$: $(u_i, u_{i+1}) \in \mathcal{F}$. A path is called *simple* if no node appears more than once on it. A state machine (SM) is a PN such that each transition has exactly one input place and one output place. A free choice (FC) net is a PN such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition.

A transition $t$ is *enabled* in a marking $M$, denoted by $M[t\rangle$, when all places in ${}^\bullet t$ are marked. An enabled transition in $M$ *fires*, removing one token from each place in ${}^\bullet t$ and adding one token to every place in $t^\bullet$. This produces a new marking $M'$ ($M[t\rangle M'$). A marking $M$ is *reachable* from $M_o$ if there is a sequence of *firings* $t_1 t_2 \cdots t_n$ that transforms $M_o$ into $M$ ($M_o[t_1 t_2 \cdots t_n\rangle M$); hence $t_1 t_2 \cdots t_n$ is a *feasible* sequence. The set of reachable markings from $M_o$ is denoted by $[M_o\rangle$. The graphical representation of a reachability set with the vertices corresponding to markings and arcs corresponding to transitions between markings is called a *reachability graph* (RG). Two transitions $t_1$ and $t_2$ are concurrent if there exists a marking in which both transitions are enabled and the firing of $t_1$ or $t_2$ does not disable the other.

A PN is *live* if every transition can be infinitely enabled through some *feasible* sequence of *firings* from any marking in $[M_o\rangle$. A PN is *safe* if no marking in $[M_o\rangle$ can assign more than one token to any place. A place is *redundant* if its removal preserves the set of feasible sequences in the PN. In the sequel, we will assume that all the considered PN's are *free choice*, *live*, *safe*, and do not contain *redundant* places.[1]

A PN can be decomposed into a potentially exponential set of strongly connected state machines, also named SM-components (SM's) [11]. In particular, live and safe free-choice PN's are covered by *one-token* SM's; that is, SM's that contain exactly one token [11]. Computing SM's is reduced to solving a linear programming model, with polynomial complexity [18]. An SM-*cover* (SMC) is a subset of *one-token* SM's such that every place in a PN is included at least in one SM.

An STG is a triple $D = \langle \Sigma, A, \Lambda \rangle$, where $\Sigma$ is its underlying PN, $A$ is a set of input $A_I$ and output $A_O$ signals, and $\Lambda$ is a labeling function $\Lambda: \mathcal{T} \rightarrow A \times \{+, -\}$, in which the transitions are interpreted as value changes on circuit signals. Rising and falling transitions of a signal $a \in A$ are denoted by $a+$ and $a-$, respectively, while $a*$ denotes a generic rising or falling transition. Multiple transitions for a signal will be distinguished by means of indexes, e.g., $a_1+$, $a_2+$. (In figures, instead of indexes for $a_{1^+}$, $a_{+/1}$ will be used.) An STG is *autoconcurrent* if it contains a pair of concurrent transitions of the same signal.

An STG is graphically represented as a directed graph with transitions denoted by their names and places by circles, where places that have only one transition in its preset and postset are usually omitted. Also, transitions of input signals are underlined. Fig. 1(a) depicts a free-choice STG, taken from [19], that will be used throughout this work. The example contains input ($A_I = \{a, b\}$) and output ($A_O = \{c, d\}$) signals. The corresponding reachability graph of the STG is depicted in Fig. 1(b). Fig. 2 depicts three SM's that cover the STG.

[1] Checking for liveness, safeness, and redundant places can be done in polynomial time for FC nets [12].

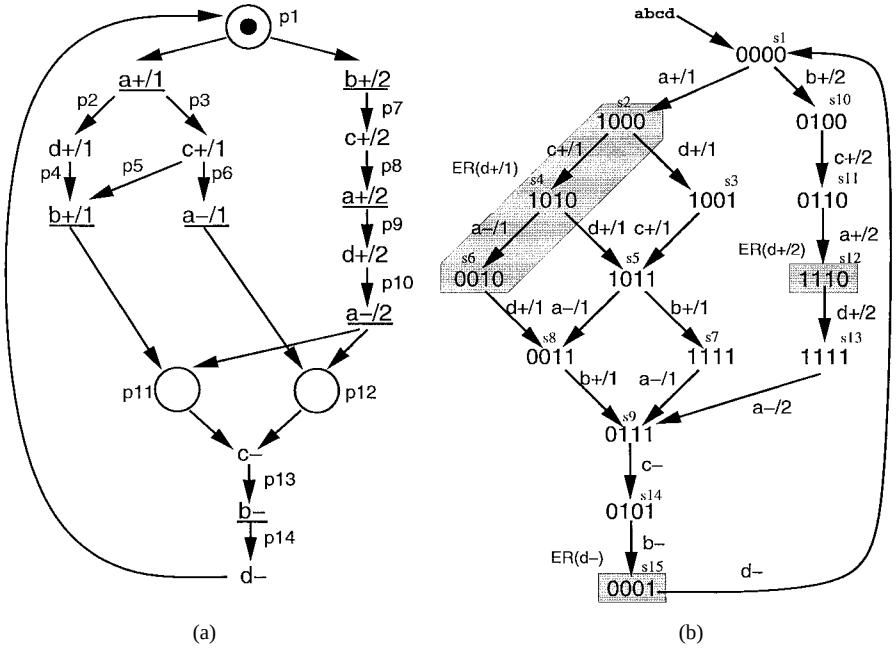



Fig. 1. (a) STG example and (b) corresponding reachability graph.

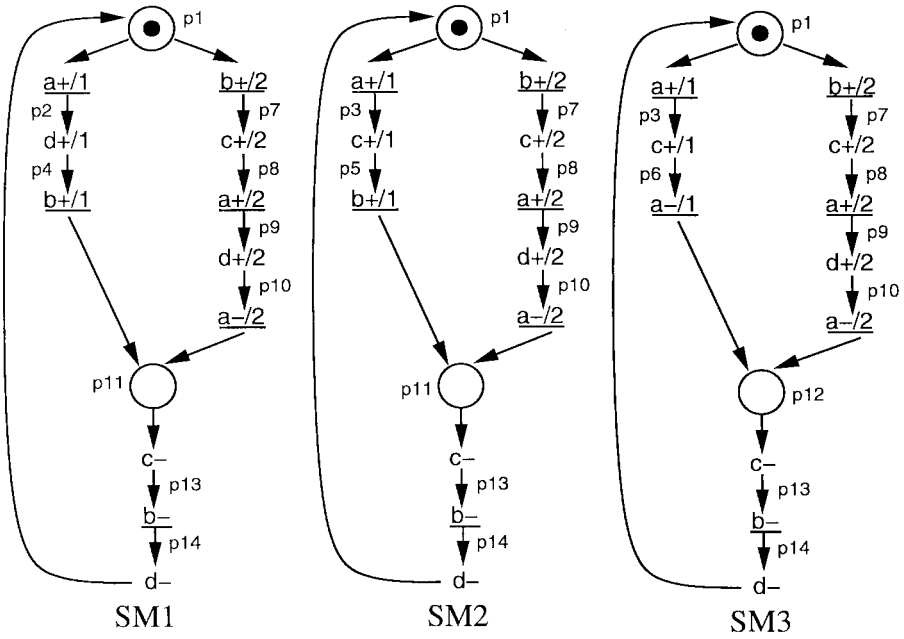



Fig. 2. SM-components of example depicted in Fig. 1(a).

Each marking of an STG is encoded with a *binary code* of signal values by means of a labeling function $\lambda: [M_o\rangle \rightarrow \{0, 1\}^{|A|}$, where $\lambda(M)_a$ denotes the binary value for signal $a$. The function $\lambda$ must *consistently encode* the STG markings; that is, no marking $M$ can have an enabled rising (falling) transition $M[a+\rangle$ $(M[a-\rangle)$ if $\lambda(M)_a = 1 (\lambda(M)_a = 0)$.

In a *nonautoconcurrent* STG, transition $a_{i^*}$ is a *predecessor* of $a_{j^*}$ if there exists a feasible sequence $a_{i^*}\sigma a_{j^*}$ that does not include other transitions of signal $a$. Conversely, $a_{j^*}$ is a *successor* of $a_{i^*}$—we will also say that the pair $(a_i*, a_{j^*})$ is *adjacent*. The set of predecessors (successors) of $a_{i^*}$ is denoted by prev$(a_{i^*})$ (next$(a_{i^*})$). In Fig. 1(b), transition $d-$ has two successor transitions $d+/1$ and $d+/2$, while at the same time $d-$ is a single predecessor to both rising transitions.

An STG is called *output semimodular* if no output signal transition $a_{i^*}$ enabled at any reachable marking can be disabled by the transition of another signal $b_{j^*}$ [20]. If an STG is output semimodular, then it can be implemented without producing unspecified changes of the output signals; that is, without introducing *hazards*.

### C. Signal Regions

To derive the correspondence among the signal transitions, the reachable markings, and the properties of the specification, different *signal regions* are defined.

TABLE I
SIGNAL REGIONS FOR THE EXAMPLE IN FIG. 1

| | d+/1 | d+/2 | d− |
|---|---|---|---|
| ER | $p_4p_{11}p_{13}p_{14}$ | $p_{10}p_{11}p_{12}p_{13}p_{14}$ | $p_1p_2p_7p_8p_9$ |
| $\widehat{\text{ER}}$ | -0-0 | 1110 | 0001 |
| $\widehat{\text{QR}}$ | -0-1 + -111 + 0101 | -111 + 0-1- + 0101 | 0-00 + 01-0 |

The *excitation region* $\text{ER}(a_{i^*})$ is the set of markings in which transition $a_{i^*}$ is enabled. It can be shown that, for live and safe free-choice STG, excitation regions are connected sets of markings. The *quiescent region* $\text{QR}(a_{i^*})$ is the maximal set of markings that are reached from $\text{ER}(a_{i^*})$ after firing $a_{i^*}$ without enabling any other transition $a_{j^*}$. The *restricted quiescent region* $\text{QR}^r(a_{i^*})$ is the subset of the quiescent region $\text{QR}(a_{i^*})$ that does not contain markings of other QR's of signal $a$.

The *generalized rising (falling) excitation region* of signal $a$ is the union of all excitation regions $\text{ER}(a_{i^+})$ ($\text{ER}(a_{i^-})$), denoted by $\text{GER}(a+)$ and $\text{GER}(a-)$. The *generalized one (zero) quiescent region* of $a$ is the union of all quiescent regions $\text{QR}(a_{i^+})$ ($\text{QR}(a_{i^-})$), and it is denoted by $\text{GQR}(a+)$ ($\text{GQR}(a-)$). Fig. 1(b) depicts the excitation regions $\text{ER}(d+/1)$, $\text{ER}(d+/2)$, $\text{ER}(d-)$ for the output signal $d$.

Regions are collections of markings; hence, we use the operator $\widehat{\ }$ to define the characteristic function of the binary codes of the markings in a set or region. Additionally, we will define the dc-*set* as the set of nonused binary codes, i.e., $\{0,1\}^{|A|} - \widehat{[M_o\rangle}$. Examples of other regions and binary codes for signal $d$ can be found in Table I.

### D. State Coding

An STG is said to satisfy the *complete state coding* (CSC) property if, when the same binary code is assigned to two different markings, the output signals enabled at both markings are identical, i.e., $\forall M_1, M_2 \in [M_o\rangle$: $\lambda(M_1) = \lambda(M_2) \Rightarrow (\forall a \in A_O, \exists a_{i^*}, a_j*: M_1[a_{i^*}\rangle \Leftrightarrow M_2[a_{j^*}\rangle)$. An efficient technique to verify the CSC property can be derived if instead of analyzing individual markings, the encoding properties are checked in terms of sets of markings related to the structure of the STG, i.e., $\forall a \in A_O$: $\widehat{\text{GER}}(a+) \cdot \widehat{\text{GQR}}(a-) = \emptyset \wedge \widehat{\text{GER}}(a-) \cdot \widehat{\text{GQR}}(a+) = \emptyset$.

A more restrictive property, the *unique state coding* (USC) condition, holds if all reachable markings of the STG are assigned a unique binary code, i.e., $\forall M_1, M_2 \in [M_o\rangle$: $M_1 \neq M_2 \Rightarrow \lambda(M_1) \neq \lambda(M_2)$. The example in Fig. 1 has a USC conflict because markings $s_7$ and $s_{13}$ share the binary code (1111). However, the STG satisfies the CSC property because output transition is enabled at neither $s_7$ nor $s_{13}$ (i.e., no CSC conflict exists).

### E. Next-State Function

The derivation of a circuit that implements the behavior specified by an STG consists in finding a logic-gate realization of the next-state function for each output signal. The next-state function, $f_a$: $\{0,1\}^{|A|} \longrightarrow \{0,1\}$, of a signal $a \in A_O$ is defined as follows [21]:

$$f_a(v) = \begin{cases} 1, & \text{if } v \in \widehat{\text{GER}}(a+) \cup \widehat{\text{GQR}}(a+) \\ 0, & \text{if } v \in \widehat{\text{GER}}(a-) \cup \widehat{\text{GQR}}(a-) \\ -, & \text{otherwise.} \end{cases}$$

For any STG that fulfills the consistency and CSC conditions, $f_a$ is consistently defined, i.e., $\{\text{on}(f_a), \text{off}(f_a), \text{dc}(f_a)\}$ is a complete partition of $\{0,1\}^{|A|}$. Note that for any pair of output signals $a$ and $b$, $\text{dc}(f_a) = \text{dc}(f_b) = \text{DC}$, where DC denotes the dc-*set* of the reachability graph.

An implementation of the next-state function $f_a$ by a cover $\mathcal{C}$ is correct if

$$\text{on}(f_a) \subseteq \mathcal{C} \subseteq \text{on}(f_a) \cup \text{DC}. \tag{1}$$

## III. SPEED-INDEPENDENCE SYNTHESIS CONDITIONS

The derivation of an SI circuit from an STG specification requires two types of correctness conditions [20].

- *Specification correctness conditions:* Consistency, output semimodularity, and CSC. These conditions have been defined in Section II and guarantee that a correct SI circuit can be derived from the STG specification.
- *Implementation correctness conditions:* These conditions guarantee that a given circuit implements the desired behavior. We can distinguish two types of conditions.
  - — Correct next-state function condition (1).
  - — Conditions for hazard freeness, which depend on the specific circuit architecture chosen for the implementation. These conditions will be discussed in Section III-B.

Consistency and CSC are necessary and sufficient conditions for the existence of a consistent next-state function. Output semimodularity is a necessary condition for the existence of a hazard-free implementation of the behavior. In the case where all next-state functions can be correctly implemented by a hazard-free complex gate, the circuit is guaranteed to be SI [5]. The implementability conditions of SI circuits have been exhaustively investigated in [7], [17], [19], and [22].

However, it is not always possible to implement each next-state function with one complex gate. In general, gate libraries impose constraints on the size and functionality of the logic functions that can be implemented with only one gate.

This section first introduces three different implementation architectures and discusses sufficient conditions for obtaining correct implementations of the next-state functions. It is shown that these conditions can be formulated in terms of requirements for the covers of the corresponding signal regions. The rest of the section is devoted to discussing the conditions for hazard freeness that guarantee the synthesis of an SI circuit.

One of the architectures is chosen for the illustration of the methodology of structural synthesis throughout the paper. However, the suggested methods are easily adapted to other architecture styles as well.

### A. Implementation Architectures

*1) Atomic Complex Gate Per Signal:* This is the initial architecture for SI circuits studied in [5] and [23]. The circuit

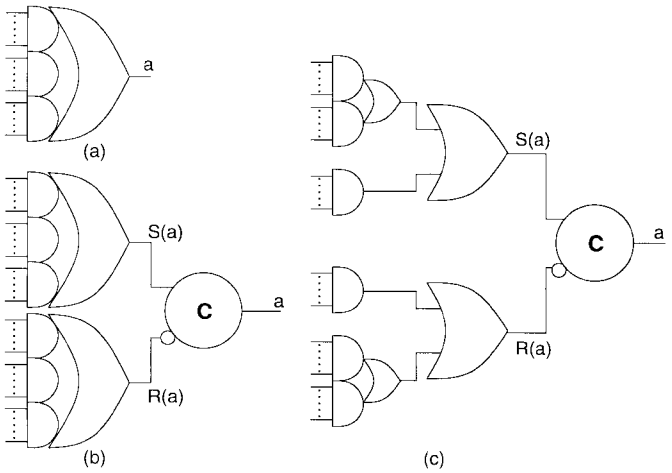



Fig. 3. Implementation architectures.

is implemented as a network of atomic gates, each one implementing one output signal. The Boolean function for each gate can be represented as a sum of products (SOP). A simple example of such gate is presented in Fig. 3(a). Each atomic gate contains a combinational part and a possibly sequential part implemented as an internal feedback. The delay between its "ANDing" and "ORing" nodes and the internal feedback is assumed to be negligible. In the figures, the gate representation is used to denote the implemented logic function, but the actual implementation is resolved on the transistor level.

The circuit is assumed to be derived by building a correct cover $C$ for $f_a$ [according to (1)] and implemented by a single complex gate. It was shown in [5] that for correct STG's, this equation gives necessary and sufficient conditions for the speed independence of the implementation (i.e., no additional architecture-specific conditions are needed). However, the requirement to implement each cover by a single gate might be quite unrealistic in practice, which is the weakest point for this approach.

*2) Atomic Complex Gate Per Excitation Function:* This architecture was suggested and studied extensively in a number of papers, e.g., [20] and [24]. It assumes that a separate memory element is used to produce an output signal. The set $(S(a))$ and reset $(R(a))$ *excitation functions* for signal $a$ are fed to the memory element. They are implemented as atomic complex gates. Fig. 3(b) shows an example of such architecture with a C-latch used as a memory element.

Sufficient conditions that guarantee the implementation correctness of the next-state function are the following:

$$\widehat{\mathrm{GER}}(a+) \subseteq S(a) \subseteq \mathrm{on}(f_a) \cup DC \wedge$$
$$\widehat{\mathrm{GER}}(a-) \subseteq R(a) \subseteq \mathrm{off}(f_a) \cup DC. \qquad (2)$$

The set function for signal $a$ must be turned on every time some rising transition $a_i+$ is enabled and turned off before the enabling of any falling transition $a_j-$; similarly for the reset function. However, the conditions in (2) do not guarantee an SI circuit. Sufficient extra conditions for hazard freeness will be discussed in Section III-B. It is possible to show the existence of an implementation in this architecture for any STG satisfying the CSC condition [5].

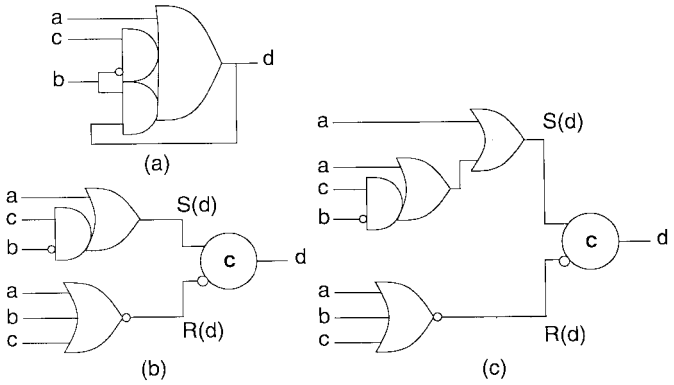



Fig. 4. Three speed-independent implementations for signal $d$.

*3) Atomic Complex Gate Per Excitation Region:* Signals in this architecture are created using networks of atomic complex gates to implement the set and reset functions of the memory element. Each transition is implemented by a single gate, which is then connected to an OR-gate whose output is in turn fed into the memory element. As a result, smaller complex gates are used. The basic structure of this architecture is shown in Fig. 3(c).

In this architecture, every gate $C(a_{i+})$ at the first level of the set function implements the behavior of a single rising transition $a_{i+}$. This gate must be turned on every time transition $a_{i+}$ is enabled and turned off before the enabling of any falling transition $a_j-$; similarly for the reset function.

In a nonautoconcurrent STG, only one transition of the signal can be enabled at a certain instant. Therefore, the proposed architecture evolves under a *one-hot encoding* discipline of the gates at the first level of the set and reset networks. Only one of the gates can be "ON" at the same time, being responsible for the output signal to switch. The rising and falling signal switching is produced due to the alternate activation of set and reset networks.

The implementation correctness condition for the covers $C(a_{i^*})$ is similar to condition (2) but is limited to only use its excitation and quiescent region

$$\widehat{\mathrm{ER}}(a_{i^*}) \subseteq C(a_{i^*}) \subseteq \widehat{\mathrm{ER}}(a_{i^*}) \cup \widehat{\mathrm{QR}}(a_{i^*}) \cup \mathrm{DC}. \qquad (3)$$

The detailed discussion on the sufficient conditions to ensure an SI implementation with this architecture can found in [7] and [19]. A general discussion on these conditions is presented in Section III-B.

Fig. 4 shows the implementations of signal $d$ from the STG in Fig. 1 in all three architectures.

More recent developments aim at the decomposition of complex gates used to implement each excitation region. The goal of these techniques is to guarantee the implementability of the circuit in a particular gate library or with a network of two-input gates [25], [26].

From the review of the possible architectures, we can conclude that the *architecture-specific* conditions for correct implementations can always be formulated in terms of covering the signal regions. In Section VI, it will be shown how to obtain approximations for each signal region by using the information contained in the structure of the STG rather than its RG. Therefore, the synthesis techniques suggested in

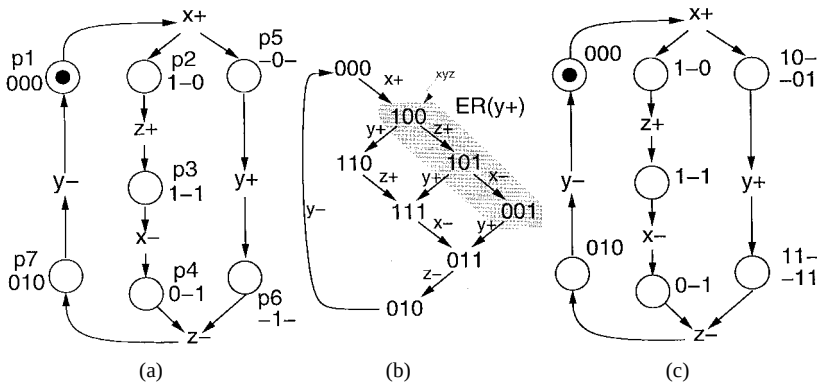



Fig. 5. (a) STG and covering cubes for places, (b) reachability graph, and (c) refined covers.

this work can be adapted to any implementation architecture. Further, we will illustrate the synthesis method in application to the architecture in Fig. 3(b), when the set and reset functions are implemented as atomic complex gates. Note, however, that there are no strict borders between different architecture styles and, for optimization purposes, we can easily admit the implementation of one signal of a circuit as an atomic complex gate while the other is implemented by the set and reset networks. These issues are mainly addressed in Section VIII, where the circuit minimization loop is discussed.

### B. Conditions for Hazard Freeness

In the previous section, we introduced three main types of implementation architectures and formulated the conditions that must be satisfied by Boolean functions of gates to ensure the proper values of implemented signals. However, this functional correctness is not sufficient to guarantee the hazard-free behavior of a circuit. Even when the Boolean functions of gates are defined according to the requirements of Section III-A, the behavior of the circuit can be hazardous due to the delays in the propagation of signals through the gates. This must be avoided in speed-independent designs. In this section, we introduce the sufficient conditions that will capture the absence of hazards during the operation of a circuit.

From now on, unless it is pointed out explicitly, we assume that each output signal of the STG is implemented by complex gates for set and reset functions—atomic complex gate per excitation function—with a C-latch as memory element.

The correctness of the set and reset covers is not sufficient to guarantee the SI behavior of the implementations. Additionally, these covers must be *monotonic*. Intuitively, $S(a)$ $(R(a))$ is said to be *monotonic* if it changes exactly twice in any sequence of firing transitions, rising at a marking in $\mathrm{GER}(a+)$ $(\mathrm{GER}(a-))$ and falling either inside $\mathrm{GQR}(a+)$ $(\mathrm{GQR}(a-))$ or before entering $\mathrm{GER}(a-)$ $(\mathrm{GER}(a+))$.

For example, (see Fig. 1), assume that $R(d)$ covers markings $s_{15}$ and $s_{10}$. $R(d)$ is correct, but if the circuit follows the sequence $s_{15} \rightarrow s_1 \rightarrow s_{10} \cdots$, it might produce an undesired $1 \rightarrow 0 \rightarrow 1$ glitch at function $R(d)$ that might eventually be propagated to output $d$.

The following property describes how the set and reset covers can be verified to be monotonic exploring the reachable markings of the STG rather than its feasible firing sequences.

*Property 1 [Monotonic Covers]:* A set function $S(a)$ is said to be *monotonic* iff $\forall M \in \mathrm{GQR}(a+)$ such that its code $\lambda(M)$ is covered by $S(a)$, then $\forall M' \in \mathrm{GQR}(a+)$: $M'[t\rangle M$, the binary code $\lambda(M')$ is also covered by $S(a)$. A reset function $R(a)$ is said to be *monotonic* iff $\forall M \in \mathrm{GQR}(a-)$ such that its code $\lambda(M)$ is covered by $R(a)$, then $\forall M' \in \mathrm{GQR}(a-)$: $M'[t\rangle M$, the binary code $\lambda(M')$ is also covered by $R(a)$.

For the particular case of the atomic complex gate per excitation region architecture, each cover $C(a_{i^*})$ must satisfy an additional monotonic condition designed to guarantee a hazard-free alternating one-hot activation of set and reset networks. A cover $C(a_{i^*})$ cannot freely use its QR as dc-*set* because some of its markings may be shared by other covers for signal $a$. In Fig. 1, marking $s_9$ is shared in the QR's of both transitions $d+/1$ and $d+/2$. If the cover $C(d+/1)$ includes that shared marking, both covers $C(d+/1)$ and $C(d+/2)$ will be incorrectly excited (not necessarily at the same time) whenever transition $d+/2$ is expected to be fired. The additional condition to guarantee the monotonic alternating activation of set and reset networks can be expressed by using the *restricted quiescent region* as

$$\widehat{\mathrm{ER}}(a_{i^*}) \subseteq C(a_{i^*}) \subseteq \widehat{\mathrm{ER}}(a_{i^*}) \cup \widehat{\mathrm{QR}}^{r}(a_{i^*}) \cup \mathrm{DC}. \quad (4)$$

Imposing restrictions on the markings that can be covered to guarantee the one-hot enabling discipline is equivalent to the *single entrance constraint* described by other authors [7], [24]. However, in order to verify this restriction, restricted quiescent regions are easier to build and structurally characterize than firing sequences.

The result proved in [7] and [19] is the following: *"If the correct set and reset covers satisfy the monotonicity conditions, the circuit implementation is speed independent."* The main purpose of the following sections is to show how the correctness and monotonicity conditions can be ensured for the set and reset covers without generating the reachability graph of the STG.

## IV. Applying Structural Methods to Synthesis

This section gives an intuitive picture of the proposed structural methods by using the example depicted in Fig. 5(a). The techniques here described are fundamental to support the overall synthesis process keeping its complexity polynomial.

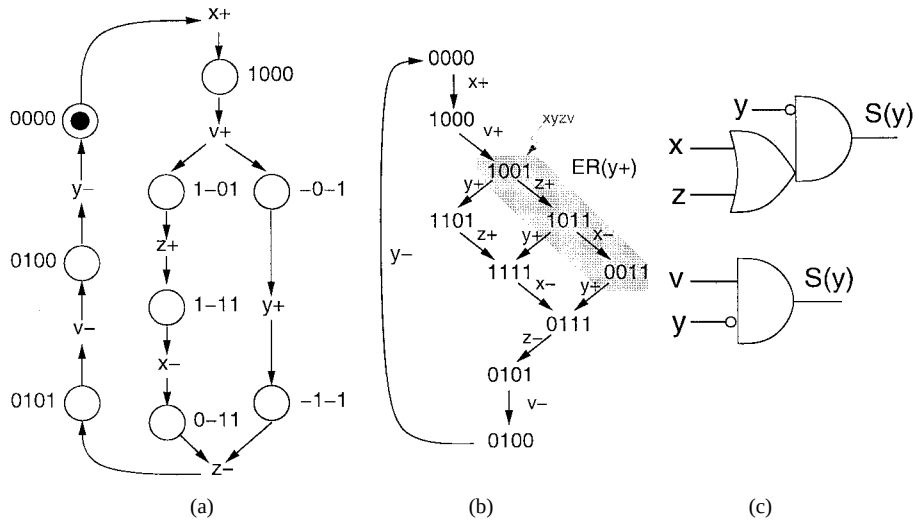



Fig. 6. Signal insertion. (a) STG and covering cubes for places, (b) reachability graph, and (c) implementation for signal $y$.

Let us assume that we wish to derive a logic function to cover the excitation region of $y+$ [denoted ER$(y+)$]. This region corresponds to the set of markings in which place $p_5$ is marked. The encoded reachability graph obtained from the STG is depicted in Fig. 5(b), in which ER$(y+)$ is also shadowed.

By a simple structural analysis that takes polynomial time [12], we can deduce that the STG has an underlying free-choice PN in which each SM has exactly one token. We can also derive a set of SM's that cover the net (SM-cover). In this case, two SM's can be obtained, namely, the sets of nodes $\text{SM}_1 = \{p_1, x+, p_2, z+, p_3, x-, p_4, z-, p_7, y-\}$ and $\text{SM}_2 = \{p_1, x+, p_5, y+, p_6, z-, p_7, y-\}$.

Our purpose is to calculate a set of cubes that safely cover $\widehat{\text{ER}}(y+)$.[2] An initial single cube approximation can be calculated as follows. If a signal transition can fire while a given place is marked, without removing the token from the place, then the value of the signal is unknown while the place is marked. Since transitions $z+$ and $x-$ can fire when $p_5$ is marked, then the value of $x$ and $z$ is unknown in $p_5$. On the contrary, the value of $y$ can be exactly determined by analyzing the ordering relation of $y+$ and $y-$ with $p_5$. Thus, the cube $(-0-)$ can be derived for $p_5$.

However, we can easily detect that this cube $(-0-)$ is an overestimation of $\widehat{\text{ER}}(y+)$ because the binary code $(000)$ which is outside $\widehat{\text{ER}}(y+)$ is also covered. Assuming $(-0-)$ to be a cover cube for $\widehat{\text{ER}}(y+)$ leads to the erroneous conclusion on the enabling of $y+$ in $(000)$. Note that overestimation does not necessarily happen in the approximation process: for places $p_1$ and $p_7$, the cubes can be exactly calculated, i.e., $(000)$ and $(010)$, respectively. To fight with the possible overestimations, two strategies can be applied.

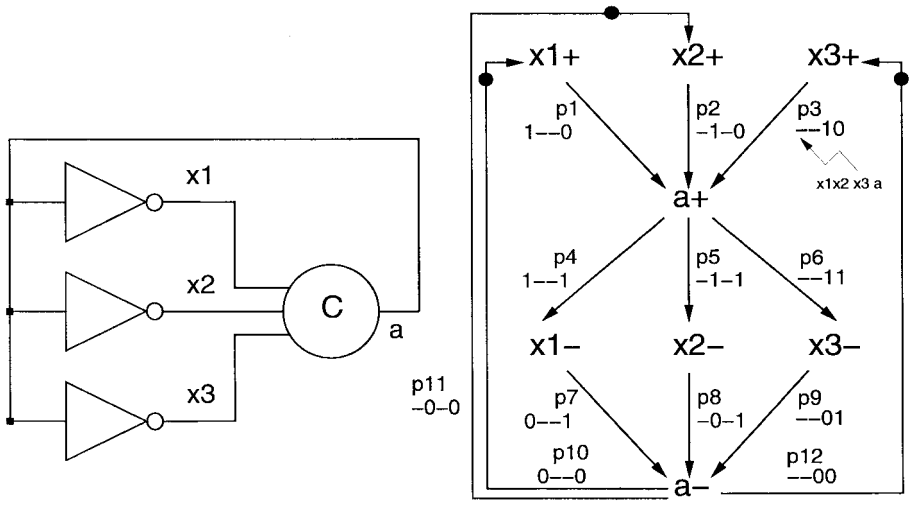
1) *Cover refinement:* Refining the place covers by analyzing the concurrent relations with other places. To obtain a multicube approximation, we use the fact that $p_5$ can only be simultaneously marked with $p_2$, $p_3$, or $p_4$. The cover of $p_5$ should be intersected with the conjunction of the covers of $p_2$, $p_3$, and $p_4$ [see Fig. 5(c)]. Then, the function $(10-) + (-01)$ correctly covers $\widehat{\text{ER}}(y+)$ [see Fig. 6(c)]. Note that, in general, several refinements may be needed.
2) *Signal insertion:* Inserting state signals in the same way as solving encoding conflicts, disambiguating covers whose intersection produces contradictions for synthesis. This is illustrated in Fig. 6, in which a new signal $v$ distinguishes the covers of $p_1$ and $p_5$. Then, the cube $(-0-1)$ correctly covers $\widehat{\text{ER}}(y+)$ [see Fig. 6(c)].

In general, both methods can be combined to obtain a correct set of covers. In this work, we only present the conditions under which a set of covers can be safely used for synthesis without the insertion of extra signals. The procedures for insertion of extra signals are covered in [27].

To give an intuitive idea about the efficiency of the structural approach, let us consider one illustrative example. Fig. 7 presents an autonomous circuit with a C-latch closed on its inputs through inverters. A C-latch is the basic cell used for the synchronization of processes in asynchronous designs. Its output rises when all its inputs are "1" and falls when all inputs are "0"; in any other case the output remains unchanged. The logic function for a C-latch is $a = x_1 \cdots x_n + a(x_1 + \cdots + x_n)$. In our example, a change on the output of the C-latch leads to a concurrent burst of input changes. The number of markings in an $n$-input circuit is $2^{n+1}$, while the number of places in the corresponding STG is only $4n$.

The use of cover cubes for the places in this example is extremely efficient because they *exactly* define the excitation regions for all signal transitions; that is, the information provided by the concurrency relations coincides with the structure of the reachability graph. Given transition $x_1-$, the cube $(1--1)$ of its predecessor place $p_4$ is an exact cover for $\widehat{\text{ER}}(x_1-)$ (signal order $x_1x_2x_a$ is used). Given transition $a+$, the intersection of cubes for its predecessor places $p_1$, $p_2$, and $p_3$ gives the single code (1110) where $a+$ is enabled $((1-0)\cdot(-1-0)\cdot(--10) = (1110))$.

[2] $\widehat{\text{ER}}(y+)$ is the set of binary codes of markings in ER$(y+)$. The cover must contain $\widehat{\text{ER}}(y+)$ (on-set) and may contain codes from the dc-*set*.




(a) (b)

Fig. 7. (a) Generalized-latch circuit and (b) its STG specification.

In this example, we have obtained the functions for signals from the structural information in the STG rather than by restoration of its reachability graph. Although the function derivation procedure is not always so simple, it allows one to present a general view of complexity reduction while using the cover cube approximations. In the rest of this paper, we describe the conditions to determine how the aforementioned covers can be iteratively improved and when the reached accuracy is sufficient to be considered correct.

## V. STG Structural Analysis

This section presents structural methods for analyzing STG's [28]. This method will be used in Section VI to find approximate covers for ER's and QR's. ER's and QR's will be approximated by a much simpler region that characterizes the markings in which a given place is marked, the so-called *marked region*. The goal of this section is to derive a single cube cover for each marked region by using a set of structural properties that can be computed in polynomial time on the size of the STG.

Based on the concurrency relations and the analysis of paths in the PN, we introduce a polynomial algorithm to verify the consistency of the STG. Consistency is a necessary condition for the synthesis of SI circuits, but it is also necessary to guarantee the existence of a consistent next-state function for the signals in the STG. Using the concurrency and the interleaving between signals, cubes will be derived to approximate the binary codes of markings in the marked regions.

### A. Concurrency Relations

The concurrency relation (CR) [5] is a conservative concept defined in terms of markings in the RG of an STG that provides a high-level view of its dynamic behavior. When two transitions can fire from a marking without disabling each other, the transitions are said to be *concurrent*. Since this is a structural property, its definition must be conservative. Two transitions may appear to be concurrent in one part of the RG while ordered in another. In that case, we should take them as concurrent because they are not always ordered.

TABLE II
$\mathcal{SCR}$ Between Signals and Places for the STG in Fig. 1

| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ | $p_{10}$ | $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | | *co* | | *co* | *co* | | | | | | *co* | | | |
| b | | | | | | *co* | | | | | | *co* | | |
| c | | *co* | | *co* | | | | | | | | | | |
| d | | | *co* | | *co* | *co* | | | | | | *co* | | |

Concurrency relations can be extended to places and signals [27]. We will refer to the formalization of concurrency between nodes and signals as signal concurrency relations (SCR).

*Definition 2 (Concurrency Relations):* The concurrency relation between pairs of nodes $(\mathcal{T} \cup \mathcal{P})$ of an STG is defined as a binary relation $\mathcal{CR}$ such that given places $p_i$, $p_j$, transitions $t_i$, $t_j$, exists $M$, $M' \in [M_o\rangle$

$$(t_i, t_j) \in \mathcal{CR} \Leftrightarrow [M[t_i t_j\rangle \wedge M[t_j t_i\rangle];$$
$$(p, t_i) \in \mathcal{CR} \Leftrightarrow [M[t_i\rangle M' \wedge M(p) > 0 \wedge M'(p) > 0];$$
$$(p_i, p_j) \in \mathcal{CR} \Leftrightarrow M(p_i) > 0 \wedge M(p_j) > 0].$$

*Definition 3 (Signal Concurrency Relations):* The signal concurrency relation between a node $u_j \in \mathcal{P} \cup \mathcal{T}$ and a signal $a \in A$ is defined as a binary relation $\mathcal{SCR}$ such that $(u_j, a) \in \mathcal{SCR} \Leftrightarrow \exists a_{i^*}\colon (u_j, a_{i^*}) \in \mathcal{CR}$.

Polynomial algorithms for the computation of the concurrency relations of a live and safe free-choice PN have been presented in [29]. As an example, Table II depicts the $\mathcal{SCR}$'s for the places of STG in Fig. 1(a) [where *co* indicates those pairs $(p_i, a)$ that are concurrent].

### B. Consistency Verification

If an STG is not consistent, it cannot be implemented by a logic circuit. Therefore, consistency must be checked before performing the synthesis step. This section presents an efficient algorithm to verify the consistency of a live, safe, and irredundant free-choice STG by using the concurrency relations and the structure of the underlying PN.

An STG satisfies the *consistency* condition if it does not contain *autoconcurrent* transitions and every sequence of signal transitions is *switchover correct* [20]. To avoid autoconcurrent transitions, no pair $a_{i^*}$ and $a_{j^*}$ of transitions of the same signal is allowed to be simultaneously enabled at the same marking. Switchover correctness requires the value of each signal to switch from zero to one in response to a rising transition and from one to zero due to a falling transition.

Nonautoconcurrency can be structurally verified by using the signal concurrency relations, i.e., by checking that each transition $a_{i^*}$ is nonconcurrent with signal $a$ ($\forall a_{i^*}$: $(a_{i^*}, a) \notin \mathcal{SCR}$).

The switchover correctness of a nonautoconcurrent STG can be verified by checking that all adjacent transitions of the same signal have alternating switching directions. A pair of transitions of the same signal can be determined to be adjacent by finding a particular path in the STG connecting both transitions. The following property characterizes the relation between the formal definition of adjacency on the RG and its efficient computation on the structure of the STG.

*Property 4 (Structural Characterization of Adjacency) [Necessary Condition]:* In a live and free-choice STG, a transition $a_{j^*} \in \text{next}(a_{i^*})$ if there is a simple path $L$ between $a_{i^*}$ and $a_{j^*}$ such that:

1) no place $p \in L$ is concurrent to signal $a$;
2) $L$ contains no other transitions of signal $a$ except $a_{i^*}$ and $a_{j^*}$.

*Proof:* The proof is done by induction on the length of the path $L$. (The length of the path is always odd.)

1) $|L| = 1$. Then $a_{i^*} \rightarrow p \rightarrow a_{j^*}$ (where $\rightarrow$ denote arcs between STG nodes). If $p$ is a choice place, then it is free choice and the sequence $a_{i^*}$, $a_{j^*}$ is feasible. If $p$ is not a choice place, then the token in $p$ can be consumed only by transition $a_{j^*}$. From the liveness of the STG, it follows that there exists a feasible sequence that contains both $a_{i^*}$ and $a_{j^*}$. Suppose that in any such sequence there is some other transition $a_{k^*}$ between $a_{i^*}$ and $a_{j^*}$. Clearly, $p$ is concurrent to any transition between $a_{i^*}$ and $a_{j^*}$, and so it is concurrent to $a_{k^*}$, which contradicts the initial assumption. Therefore, $a_{j^*} \in \text{next}(a_{i^*})$.
2) From the statement's being true for $|L| = n$, it follows that it is also true for $|L| = n + 2$. Consider the last transition $b_{j^*} \in L$ before $a_{j^*}$, i.e., $b_{j^*} \rightarrow p \rightarrow a_{j^*}$. The path $L' \subset L$ between $a_{i^*}$ and $b_{j^*}$ has length $n$, and by the induction assumption, there exists feasible sequence $a_{i^*}$, $\sigma$, $b_{j^*}$ such that $\sigma$ does not contain any transition of signal $a$. Let us show that $\sigma$ can be extended as $a_{i^*}$, $\sigma$, $\delta$, $a_{j^*}$, where $\delta$ contains no transitions of signal $a$. This clearly follows from the consideration of item 1) for $b_{j^*} \rightarrow p \rightarrow a_{j^*}$, and therefore, $a_{j^*} \in \text{next}(a_{i^*})$. ■

Assuming that an STG is nonautoconcurrent, Property 4 provides the necessary conditions to check whether $a_{i^*}$ is adjacent to $a_{j^*}$. To derive the sufficient conditions, we need to introduce several additional notions.

A path $L$ starting at $a_{i^*}$ and ending at $b_{j^*}$ is called *realizable* by a feasible sequence $a_{i^*}$, $\sigma$, $b_{j^*}$ if the sequence $a_{i^*}$, $\sigma$, $b_{j^*}$ includes all transitions in $L$. The reason to introduce the realizable paths is to restrict the number of simple paths to be analyzed when constructing the set next($a_{i^*}$). Actually, it is sufficient to consider only those simple paths that are realizable by $a_{i^*}$, $\sigma$, $b_{j^*}$, where $\sigma$ contains no transition of signal $a$.

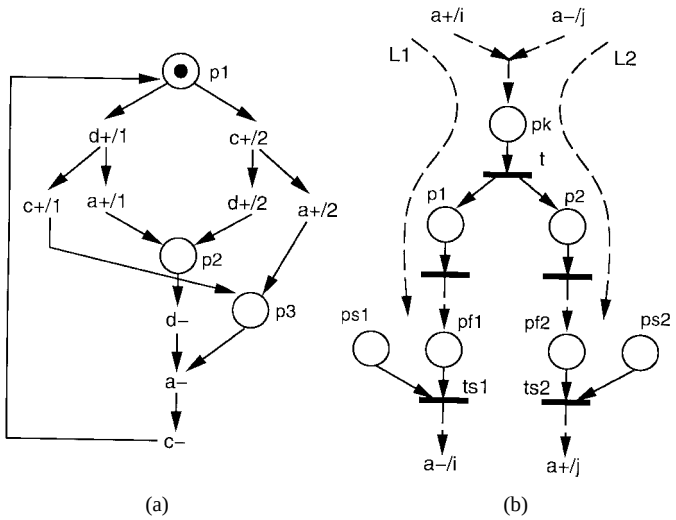



Fig. 8. STG showing (a) insufficiency of Property 4 and (b) nonconsistently interleaved place $p_k$.

The necessary conditions that characterize the paths between adjacent transitions of the same signal (given by Property 4) require any place in the path to be nonconcurrent to all transitions of the considered signal. This condition is not sufficient, as can be seen from the example in Fig. 8(a). In this STG, the sequence $d+/1$, $a+/1$, $d-$, $c+/1$, $a-$ is feasible, and therefore $a- \in \text{next}(a+/1)$. However, place $p_2$ is concurrent to $a+/2$, and the only simple path $L$ between $a+/1$ and $a-$ goes through $p_2$.

To obtain sufficient conditions for the adjacency between transitions of the same signal, it is necessary to distinguish which concurrency relations are not relevant for adjacency. This analysis can be done on the basis of *forward reduction* by concurrent transitions.

Informally, forward reduction of PN $\Sigma$ by a set of transitions $T1$ is obtained by removing from $\Sigma$ all the nodes starting from $T1$ that cannot be reached without the firing of some transition $t \in T1$. We will denote the resulting PN via $ForwRed(\Sigma, T1)$. The forward reduction can be obtained by the following procedure:

Remove transitions $T1$ from $\Sigma$
**do** until a fixed-point in modifying $\Sigma$ is reached
  **if** for $p$ all transitions ${}^\bullet(p)$ have been removed **then** remove $p$ from $\Sigma$
  **if** $p$ has been removed **then** remove all $t \in (p)^\bullet$.

The mechanism of forward reduction allows one to formulate the sufficient conditions for the existence of a realizable path between pairs of adjacent transitions of the same signal.

*Property 5 (Characterization of Adjacency) [Sufficient Condition]:* In a nonautoconcurrent, free-choice STG $D$, if $a_{j^*} \in \text{next}(a_{i^*})$, then there exists a simple path $L$ between $a_{i^*}$ and $a_{j^*}$ such that:

1) for $T1 = \{a_{k^*} | \exists p \in L, p || a_{k^*}\}$, transition $a_{j^*} \in ForwRed(D, T1)$;
2) $L$ contains no other transitions of signal $a$ except $a_{i^*}$ and $a_{j^*}$.

*Proof:* If condition (1) is violated for every path between $a_{i^*}$ and $a_{j^*}$, then none of the paths is realizable by a feasible sequence $a_{i^*}, \sigma, a_{j^*}$ without firing transitions of signal $a$, which are concurrent to places of the path. From this clearly follows that $a_{j^*}$ cannot be in next$(a_{i^*})$. Violation of condition (2) also trivially leads to nonexistence of $a_{i^*}, \sigma, a_{j^*}$, which realizes the path $L$, with $\sigma$ containing no transitions of signal $a$. ■

The algorithm in Fig. 9 is designed to verify the consistency of an STG and on the fly keep track of the set of adjacent transitions next$(a_{j^*})$ (based on Properties 4 and 5). The overall process is repeated for each transition. First, the nonautoconcurrency is checked by analyzing the SCR. Then, the paths in the STG are explored on testing the necessary conditions (procedure *correct_necessary*) for adjacency (see Property 4). Together with the construction of the set next, the switchover correctness is analyzed checking that all adjacent transitions have alternating switching directions. After applying the necessary conditions for searching adjacent transitions, the algorithm checks whether it guarantees the sufficiency as well. If not, the sufficient conditions are checked (procedure *correct_sufficient*) to search the remaining adjacent transitions (see Property 5). Procedures *correct_necessary* and *correct_sufficient* are similar and can be merged into one. However, we keep them separately because of the following.

- Procedure *correct_necessary* has lower complexity: $O(n^2)$ on the size of the STG against $O(n^3)$ for *correct_sufficient*;
- From our experiments with STG's from the known set of benchmarks (Section IX), the necessary conditions for adjacency always ensured the sufficiency as well. Even though the counterexamples can be easily constructed [e.g., see Fig. 8(a)], it seems that they are rarely met in practice. Therefore, most likely in checking the consistency, procedure *correct_sufficient* will not be invoked at all.

The upper bound of the complexity for both nonautoconcurrency and switchover correctness verification is $O(n^4)$ on the size of the STG (for the worst case when the necessary condition does not imply sufficiency).

Other authors have previously addressed the problem of consistency analysis using structural methods [9], [14]. However, either the obtained results are only applicable to marked graphs[3] or only sufficient conditions were proposed. To the best of our knowledge, this is the first proposed polynomial method to check consistency for any live and safe free-choice STG.

### C. Structural Approximation of the Reachability Graph

Several sets of markings, named *marked regions,* define a correspondence between the basic structural elements of an STG and its RG. This section defines this basic region, derives its fundamental properties, and shows how to approximate its binary codes by using a single cube. In the following sections, we will show how marked regions can be used to approximate the signal regions required for synthesis.

[3]Marked graphs are a subclass of free-choice Petri nets without choice places.

```
consistent_state_coding( STG, SCR ) {
 foreach (signal transition a_i*, a ∈ A) do
   //check non-autoconcurrency for a_i*
   if ((a_i*, a) ∈ SCR) then return false;
   next(a_i*) = ∅; //explore paths from a_i* to find set next
   set all nodes as non-explored;
   if (correct_neces(STG, a_i*, a_i*, SCR)) then
     D = ForwRed(STG, next(a_i*));
     if (in D there is no path between a_i* and
       transitions of a) then return(true);
     else return(correct_suf(D, a_i*, a_i*, SCR));
   else return(false);
}
correct_neces( STG, b_j*, a_i*, SCR ) {
 foreach (unexplored place p ∈ (b_j*)• : (p, a) ∉ SCR) do
   foreach (unexplored transition c_k* ∈ (p)•) do
     set c_k* explored;
     foreach (p ∈ •(c_k*)) do
       if (∀t ∈ (p)• is explored) then set p explored;
     if (c = a) then
       add c_k* to next(a_i*);
       //check switchover correctness
       if (a_i* and c_k* are both rising or falling) then
         return (false);
     else if (¬correct_neces(STG, c_k*, a_i*, SCR)) then
       return (false);
 return (true);
}
correct_suf( STG, b_j*, a_i*, SCR ) {
 foreach (unexplored place p ∈ (b_j*)•) do
   T1 = set of transitions a_m* concurrent to p;
   if (in ForwRed(STG, T1) there is no path between p and
     transitions of a) then set p explored;
   else
     foreach (unexplored transition c_k* ∈ (p)•) do
       set c_k* explored;
       foreach (p ∈ •c_k*) do
         if (∀t ∈ (p)• explored) then set p explored;
       if (c = a) then
         add c_k* to next(a_i*);
         //check switchover correctness
         if (a_i* and c_k* are both rising or falling) then
           return (false)
         else STG = ForwRed(STG, next(a_i*));
       else
       if (¬correct_suf(ForwRed(STG, T1), c_k*, a_i*, SCR)) then
         return (false);
 return (true);
}
```

Fig. 9. Structural verification algorithm for STG consistency.

*Definition 6 (Marked Region):* Given a place $p$, its marked region, denoted MR$(p)$, is the set of markings in which $p$ has at least one token, i.e., MR$(p) = \{M \in [M_o\rangle : M(p) > 0\}$.

For the example in Fig. 1(a), some marked regions are MR$(p_2) = \{p_2p_3, p_2p_5p_6, p_2p_5p_{12}\}$ and MR$(p_4) = \{p_3p_4, p_4p_5p_6, p_4p_5p_{12}\}$.

Especially useful is the combined utilization of MR's and the SM's of the STG. The whole reachability set of the STG is contained in the union of the MR's of the places in any given SM's. Additionally, if the SM satisfies the *one-token* condition, then the MR's define a total partition of the RG.

*Property 7 (Projection of the Reachability Set onto SM's):* The following properties are satisfied for any SM-component SM of a live PN.

1) The union of the marked regions of every place in SM is equivalent to the whole reachability set, i.e., $[M_o\rangle = \bigcup_{p\in \mathrm{SM}} \mathrm{MR}(p)$.
2) If SM satisfies the *one-token* condition, then the MR's of places in SM define a total partition of $[M_o\rangle$, i.e., $\forall\, p, p' \in SM$: $\mathrm{MR}(p) \cap \mathrm{MR}(p') = \emptyset$.

*Proof:*

1) In a live PN, any SM-component contains at least one token in the initial marking. Also, from its definition, any SM-component with some token in the initial marking remains marked at any reachable marking in $[M_o\rangle$. Therefore, any reachable marking marks some place in every SM-component.
2) If one SM-component satisfies the *one-token* condition, it will constantly be marked with exactly one token. Therefore, no marking can place a token on two places at the same time. ■

### D. Approximation of Marked Regions by Cover Cubes

A cover cube for an MR must cover all the markings of the region. To make the approximation more accurate, this cube should be the smallest among those possible (with the largest number of literals) [19]. Any signal that does not change in the MR of a place (is not concurrent to the place) is represented by a corresponding literal in a cover cube. The value of this signal can be determined by an interleave relation. *Interleaving* characterizes the position of a node with respect to a pair of adjacent signal transitions. For example, in Fig. 1(a), place $p_3$ is interleaved with $(a{+}/1, a{-}/1)$, whereas $p_2$ is not.

*Definition 8 (Interleave Relation):* The *interleave relation* is a binary relation $\mathcal{IR}$ between nodes in $\mathcal{P} \cup \mathcal{T}$ and pairs of adjacent transitions $a_{i^*}$ and $a_{j^*}$ of a signal $a$ such that a node $u_j$ is *interleaved* with $(a_{i^*}, a_{j^*})$ $(u_j \in \mathcal{IR}(a_{i^*}, a_{j^*}))$ if there exists a path from $a_{i^*}$ to $a_{j^*}$ containing $u_j$, which is realized by a feasible sequence $a_{i^*}, \sigma, a_{j^*}$, where $\sigma$ contains no transitions of signal $a$.

*Property 9 (Consistent Place Interleaving):* In a consistent STG, if a place $p_k$ is interleaved with a pair of adjacent transitions $(a_{i^+}, a_{i^-})$, then $p_k$ cannot be interleaved with any other adjacent pair $(a_{j^-}, a_{j^+})$, and vice versa.

*Proof:* Suppose the opposite: let place $p_k$ be interleaved with the adjacent pairs $(a_{i^+}, a_{i^-})$ and $(a_{j^-}, a_{j^+})$. By definition of interleaving, it means that there exist two simple paths $L1$ and $L2$ containing $p_k$ that are realizable by feasible sequences $a_{i^+}, \sigma_1, a_{i^-}$ and $a_{j^-}, \sigma_2, a_{j^+}$. Suppose that $p_k$ is chosen in such a way that no other place in $L1$ after $p_k$ is interleaved with any $(a_{k^-}, a_{k^+})$.

Let us consider output transitions of $p_k$.

*Case 1:* $t \in (p_k)^\bullet$, $t \in L1 \cap L2$. Then $t$ must have different output places $p_1$ and $p_2$ such that $p_1 \in L1$ and $p_2 \in L2$ (otherwise, we have contradiction to the choice of $p_k$).

*Case 2:* There exist $t_1, t_2 \in (p_k)^\bullet$ such that $t_1 \in L1$ and $t_2 \in L2$. Then $p_k$ is a free-choice place, and when it is marked in sequence $\sigma_2$, both $t_1$ and $t_2$ are enabled. Therefore, from $\sigma_2$, we can construct a new sequence $\sigma_2'$, which fires $t_1$ instead of $t_2$. Sequence $\sigma_2'$ (like $\sigma_2$) starts after firing $a_{j^-}$, and because of the consistency of the STG, the first transition of signal $a$ in $\sigma_2'$ is positive ($a_{k^+}$, for example). Now, if instead of path $L2$ we consider path $L2'$, corresponding to sequence $\sigma_2'$, we can reduce our analysis to the consideration of two paths $L1$ and $L2'$, which share not only place $p_k$ but also its output transition $t_1$ (like in Case 1).

This common case for both Cases 1 and 2 is shown in Fig. 8(b), where the dotted arcs show the considered subpaths of $L1$ and $L2$.

Given the general case in Fig. 8(b), let us consider the sequence $a_{j^-}, \sigma_2, a_{j^+}$, which realizes path $L2$. Suppose we forbid the firing of $a_{j^+}$ and instead we fire all fireable transitions from $a_{i^+}, \sigma, a_{i^-}$. Clearly, we cannot fire $a_{i^-}$ because in that case, $a_{i^-}$ will be concurrent to $a_{j^+}$, which contradicts the consistency assumption. Therefore, by firing the transitions in $L1$, we arrive at the "stop transition" (denoted by $ts1$) that cannot be fired because one of its input places is lacking necessary token [place $ps1$ in Fig. 8(b)]. Note that some other input place $pf1 \in {}^\bullet(ts1)$ is marked (also included in $L1$) due to the firing of transitions from $L1$ and because $ts1$ is the first "stop transition" in $L1$. Similarly, we can find the "stop transition" $ts2 \in L2$ with the corresponding places $ps2 \notin L2$ and $pf2 \in L2$.

Let us construct a sequence $\delta_2$ of minimal size such that:

1) the sequence $a_{j^-}, \sigma_2, a_{j^+}, \delta_2$ contains all transitions from $L1$ that follow place $p_k$ (it is always possible due to the liveness of STG);
2) each time a place in $L1$ gets a token while firing $\delta_2$, we choose a transition in $L1$ to consume this token (it is always possible because the STG is free choice).

Clearly, at some point while firing $\delta_2$, the "stop transition" $ts1$ should be released, i.e., place $ps1$ should get a token. Similarly, we can construct the feasible sequence $a_{i^+}, \sigma_1, a_{1^-}, \delta_1$ in which some transition from $\delta_1$ enables the "stop transition" $ts2$ by marking the place $ps2$.

From this follows that adding the transitions from $\delta_1$ to sequence $a_{j^-}, \sigma_2, a_{j^+}, \delta_2$ will produce the token in $ps2$ (we will denote the sequence corresponding to that via $\delta_{21}$). The preset of $ts2$ has several places, and any token in a place from the preset (being not a choice place) can be consumed only by $ts2$ itself. Therefore, while firing $\delta_{21}$, place $pf1$ should get a token as well.

Now let us start from the feasible sequence $a_{i_+}, \sigma_1, a_{1^-}$ and fire all the feasible transitions from $L2$. This process will be stopped at transition $ts2$ (according to the definition of "stop transition") when $pf2$ is marked while $ps2$ is not. If we forbid the firing of $ts2$ and will be adding the lacking transition from $\delta_{21}$, then both $ps2$ and $pf2$ will get tokens. The latter contradicts the assumption on the STG safeness. Therefore, the assumption on the consistent interleaving of $p_k$ is wrong. ■

Property 9 guarantees that if a place $p$ is nonconcurrent to signal $a$ and it is interleaved between two adjacent transitions $(a_{i^+}, a_{i^-})$ $((a_{i^-}, a_{i^+}))$, then all binary codes in $\widehat{\mathrm{MR}}(p)$ have value 1 (0) for signal $a$. This property is the basis to approximating markings by computing a single *cover cube* for each marked region.

*Lemma 10 [19] (Cover Cube for MR's):* The *cover cube* $C_p$ for $\mathrm{MR}(p)$ is the smallest cube that covers $\widehat{\mathrm{MR}}(p)$ such

TABLE III
COVER CUBES FOR THE STG IN FIG. 1 (SIGNAL ORDER $a$, $b$, $c$, $d$)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\mathcal{C}_{p_1}$ | 0000 | $\mathcal{C}_{p_2}$ | -0-0 | $\mathcal{C}_{p_3}$ | 100- | $\mathcal{C}_{p_4}$ | -0-1 |
| $\mathcal{C}_{p_5}$ | -01- | $\mathcal{C}_{p_6}$ | 1-1- | $\mathcal{C}_{p_7}$ | 0100 | $\mathcal{C}_{p_8}$ | 0110 |
| $\mathcal{C}_{p_9}$ | 1110 | $\mathcal{C}_{p_{10}}$ | 1111 | $\mathcal{C}_{p_{11}}$ | -111 | $\mathcal{C}_{p_{12}}$ | 0-1- |
| $\mathcal{C}_{p_{13}}$ | 0101 | $\mathcal{C}_{p_{14}}$ | 0001 | | | | |

that for every signal $b$:

1) if $b$ is nonconcurrent to $p$ ($(p, b) \notin \mathcal{SCR}$), then

$$\mathcal{C}_p^b = \begin{cases} 0, & \text{if } b = 0 \text{ in } \widehat{\text{MR}}(p) \\ 1, & \text{if } b = 1 \text{ in } \widehat{\text{MR}}(p); \end{cases}$$

2) if $b$ is concurrent to $p$, then $\mathcal{C}_p^b = -$;

where $\mathcal{C}^b$ indicates the $b$th component bit of $\mathcal{C}$.Given a place $p$, a literal must appear in the cube for any nonconcurrent signal $a$ to $p$ ($(a, p) \notin \mathcal{SCR}$). For any arbitrary place $p$, the value of the signal $a$ in a corresponding cover cube is determined by checking if $p$ is interleaved between pairs of adjacent rise–fall or fall–rise transitions. Property 9 guarantees that the value of signal $a$ is the same for all the adjacent pairs for which $p$ is in $\mathcal{IR}$. Therefore, the *interleave relation* gives a polynomial-time algorithm (for free-choice STG's) to determine the value of literal $\mathcal{C}_p^a$

$$\mathcal{C}_p^a = \begin{cases} 1, & \text{if } \exists \text{ adjacent } (a_{i^+}, a_{i^-}) \colon p \in \mathcal{IR}(a_{i^+}, a_{i^-}) \\ 0, & \text{if } \exists \text{ adjacent } (a_{i^-}, a_{i^+}) \colon p \in \mathcal{IR}(a_{i^-}, a_{i^+}) \\ -, & \text{otherwise.} \end{cases}$$

Table III depicts the cover cubes for the places of the example in Fig. 1(a). It is also important to remark that the cover cubes are *conservative* approximations of the binary codes of the markings in MR's and that other binary codes may also be covered, e.g., codes from the dc-*set* or other regions.

## VI. STRUCTURAL APPROXIMATIONS FOR SIGNAL REGIONS

The synthesis approach for SI circuits requires an analysis of both the excitation and quiescent regions in order to check the synthesis conditions introduced in Section III (correctness and monotonicity). This section discusses a conservative technique to structurally approximate signal regions by using the marked regions of places in the STG [28]. Each signal-region approximation consists of two elements: 1) its domain in the STG, consisting of corresponding sets of places and transitions, and 2) a cover associated to each node $u$ in the domain, denoted *cover function* ($cv(u)$). However, the approximation based on concurrency relations is imprecise; it leads to the overestimation of the regions that might induce synthesis errors. Therefore, this section presents the conditions under which the covers associated to the nodes have a sufficient level of accuracy to guarantee the correctness of the synthesis. Section VII will introduce a refinement technique to increase the accuracy of the cover functions when such conditions are not satisfied. Algorithms to check the SI conditions based on these approximations will be proposed later in Section VIII.

The first part of this section shows how to obtain the set of places that constitute the domain of the approximations for GER and GQR. Then the binary codes are approximated. Initially, this is done by approximating each marked region of a place by a cover cube. Approximations might overestimate marked regions, which can be acceptable if it concerns the unreachable binary codes. An unsafe overestimation occurs when the approximation for GQR($a+$) overlaps with GER($a-$), for example. In these two regions, the implied value of the next-state function $f_a$ is different and therefore the overlapping is not acceptable. The techniques to avoid a particular case of such overlapping (due to the imprecise approximations of quiescent regions on their boundaries) are described at the end of this section. Section VII tackles the general case of the refinement of cover functions to avoid any "dangerous" overlapping.

### A. Initial Approximations

Since we have chosen the *atomic complex gate per excitation function* as the target architecture, the analysis of the SI synthesis conditions will rely on the binary codes of markings in the generalized signal regions (see Section III). Generalized signal regions are defined as a union of corresponding excitation and quiescent regions (see Section II). Therefore, they can be easily derived via approximations of ER's and QR's. Single ER's and QR's are the main objects of consideration in this section. The ground objects to express ER's and QR's are the marked region of places introduced in Section V.

An *excitation region* $\text{ER}(a_{i^*})$ corresponds to the set of markings in which transition $a_{i^*}$ is enabled. $\text{ER}(a_{i^*})$ is easily defined as the intersection of marked regions for input places of $a_{i^*}$: $\text{ER}(a_{i^*}) = \bigcap_{p \in {}^\bullet(a_{i^*})} \text{MR}(p)$. Therefore, the domain required for that region is transition $a_{i^*}$ itself. The binary codes in $\widehat{\text{ER}}(a_{i^*})$ are covered by the cover function $cv(a_{i^*})$ containing a single cube that can be directly created by intersecting the cover cubes for its predecessor places: $cv(a_{i^*}) = \bigcap_{p \in {}^\bullet(a_{i^*})} \mathcal{C}_p$.

The definition is more complex for quiescent regions. A marking is in the *quiescent region* $\text{QR}(a_{i^*})$ if it can be reached by a feasible sequence $\sigma_1 a_{i^*} \sigma_2$ such that no transition $a_{j^*} \in \text{next}(a_{i^*})$ is enabled in any prefix of $\sigma_2$[4]

$$\text{QR}(a_{i^*}) = \{M \mid \exists \sigma_1, \sigma_2 \colon M_o[\sigma_1 a_{i^*} \sigma_2\rangle M \wedge (\nexists \sigma_3 \subset \sigma_2, [\sigma_1 a_{i^*} \sigma_3 a_{j^*}\rangle \text{ is feasible}\}.$$

From this formalization and the fact that those firing sequences characterize all places interleaved between adjacent transitions $a_{i^*}$ and $a_{j^*}$ (see the results in Properties 4 and 5), the domain required to approximate $\text{QR}(a_{i^*})$ should include all places interleaved between $a_{i^*}$ and some $a_{j^*} \in \text{next}(a_{i^*})$. This domain will be denoted *quiescent place set* (QPS), i.e.,

$$\text{QPS}(a_{i^*}) = \{p \mid \exists a_{j^*} \in \text{next}(a_{i^*}) \colon p \in \mathcal{IR}(a_{i^*}, a_{j^*})\}.$$

The procedure for computing the quiescent place sets is depicted in Fig. 10. Finally, the binary codes in $\widehat{\text{QR}}(a_{i^*})$ are covered by the union of the cover functions $cv(p)$ of any place in $\text{QPS}(a_{i^*})$, where $cv(p) = \mathcal{C}_p$.

[4]That the STG is assumed to be consistent makes the existence of one sequence ($\sigma_2$) sufficient for a marking to be in $\text{QR}(a_{i^*})$.

```
compute_place_sets( STG, SCR ) {
 foreach (signal a ∈ A_O) do
   foreach (signal transition a_i*) do
     set QPS(a_i*) = ∅;
     set all nodes as non-processed;
     compute_qps( STG, SCR, a_i*, a_i* );
}
compute_qps( STG, SCR, b_j*, a_i* ) {
 foreach (non-processed p ∈ (b_j*)• : (p,a) ∉ SCR) do
   add p into QPS(a_i*);
   set p as processed;
   foreach (non-processed transition c_k* ∈ (p)•) do
     set c_k* as processed;
     if (c ≠ a) then compute_qps( STG, SCR, c_k*, a_i* );
}
```

Fig. 10. Algorithm for the efficient computation of QPS.

The proposed technique approximates the signal regions by cover functions that are initialized with the values computed for the cover cubes of places. Using cover cubes to approximate QR's immediately introduces imprecision at the boundaries of the regions. By definition, QPS($a_{i^*}$) contains all places interleaved between $a_{i^*}$ and $a_{j^*} \in$ next($a_{i^*}$). The cover function of any place $p \in {}^\bullet(a_{j^*})$ also covers some binary codes in $\widehat{\text{ER}}(a_{j^*})$; therefore, $\widehat{\text{QR}}(a_{i^*})$ is overestimated. This overestimation is unsafe because the implied value for signal $a$ in the quiescent region of $a_{i^*}$ is different from that in the excitation region of $a_{j^*}$ [note that in a consistent STG, transitions $a_{i^*}$ and $a_{j^*} \in$ next($a_{i^*}$) have different directions].

For example, in Fig. 1, place $p_{12}$ will be used to approximate the quiescent region of $c+/1$. The cover function $cv(p_{12})$ is initially built as the cover cube $\mathcal{C}_{p_{12}} = (0-1-)$. This function covers not only the binary code of $s_8 \in$ QR($c+/1$) but also $s_9 \in$ ER($c-$).

Detection of overlapping between QR and corresponding ER's leads to false negatives in the results of checking the correctness conditions. To avoid this overlapping, places $p \in {}^\bullet(a_{j^*})$ used in QPS($a_{i^*}$) should be modified into $cv(p) = cv(p) - cv(a_{j^*})$. Therefore, in the previous example, the initial cover function $cv(p_{12}) = (0-1-)$ should be modified into $cv(p_{12}) = \mathcal{C}_{p_{12}} - (\mathcal{C}_{p_{11}} \cdot \mathcal{C}_{p_{12}}) = ((0-1-) - (0111)) = ((001-) + (0-10))$; that no longer covers the binary code of $s_9$.

If, however, the *cover function* for $a_{j^*}$ is also overestimated, its deduction from marked regions of places from ${}^\bullet(a_{j^*})$ may result in an *underestimation* of the cover functions for these places. Let us assume that there exists a place $p \in {}^\bullet(a_{j^*})$ and a marking $M \in$ MR($p$), but $M \notin$ MR($a_{j^*}$). If $\lambda(M) \sqsubset \mathcal{C}_{a_{j^*}}$, the computation of $cv(p) - cv(a_{j^*})$ will incorrectly eliminate $M$ from the quiescent region approximation. Underestimation of quiescent regions is dangerous because the correctness of covers for the set and reset functions is checked by ER's and QR's, and in case of underestimation, the result of the check might be erroneous (this is the source of false positives in the correctness check). In particular, if the synthesis algorithm detects that all markings in a GQR are covered, it may incorrectly decide to eliminate the memory element (see Sections III and VIII).

The domains of the excitation and quiescent regions, as well as the cover functions for both places and transitions for signal $d$ (see Fig. 1), are depicted in Table IV.

TABLE IV
SIGNAL REGION APPROXIMATIONS FOR THE STG IN FIG. 1

| | d+/1 | d+/2 | d− |
|---|---|---|---|
| ER | $\{s_2, s_4, s_6\}$ | $\{s_{12}\}$ | $\{s_{15}\}$ |
| QR | $\{s_3, s_5, s_7, s_8, s_9, s_{14}\}$ | $\{s_9, s_{13}, s_{14}\}$ | $\{s_1, s_{10}, s_{11}\}$ |
| $QR'$ | $\{s_3, s_5, s_7, s_8\}$ | $\{s_{13}\}$ | $\{s_1, s_{10}, s_{11}\}$ |
| $\widehat{\text{ER}}$ | 10-0 + -010 | 1110 | 0001 |
| $\widehat{\text{QR}}$ | 10-1 + --11 + 01-1 | -111 + 01-1 | 0-00 + 01-0 |
| $\widehat{\text{QR}}'$ | 10-1 + -011 + 1-11 | 1111 | 0-00 + 01-0 |

### B. Correctness of the Signal Region Covers

Each one of the nodes used in the domain of the structural approximations has been assigned a logic function, named *cover function*. This function is designed to approximate the binary codes of the markings in the considered signal region. In general, the complexity of the cover function is directly related to the accuracy of the approximation. Single cubes are compact approximations but may overestimate the region. More complex functions are less compact but have better accuracy. In any case, overestimating the regions must be avoided because it may lead to incorrect synthesis.

Property 7 provides the structural information to detect those cover functions that are overestimated. This result indicates that every reachable marking $M$ is projected into any SM-component of an STG, and in particular, its binary code $\lambda(M)$ will be covered by at least one cover cube of a place in every SM-component.

Let us assume that an STG satisfies the USC condition; that is, no pair of markings share the same binary code. Now, let us also assume that we have a one-token SM-component of the STG in which the intersection of the cover cubes $\mathcal{C}_{p_i}$ and $\mathcal{C}_{p_j}$ for two of its places ($p_i$ and $p_j$) is nonempty. In principle. this intersection contradicts the USC assumption, since the same marking cannot be projected into MR($p_i$) and MR($p_j$) at the same time (see the proof of Property 7). The only reason to have such an intersection is if one (or both) of the cover cubes overestimates the binary codes in its marked regions.

The accuracy of the cover functions can be verified by checking the intersection between cover cubes $\mathcal{C}_{p_i}$ and $\mathcal{C}_{p_j}$ for all pairs of places $(p_i, p_j)$ from the same SM-component of an SM-cover. An STG satisfying the empty intersection for all pairs of places within every SM-component in an SM-cover is said to be *free of structural coding conflicts* [27], [30].

*Definition 11 (Structural Coding Conflicts):* Given an SM-cover SMC, the STG is said to be free of structural coding conflicts if for all SM $\in$ SMC the intersection of the cover cubes for any pair of different places in the SM is empty, i.e., $\forall$ SM $\in$ SMC: $[\forall (p_i, p_j) \in \text{SM}, i \neq j: \mathcal{C}_{p_i} \cdot \mathcal{C}_{p_j} = \emptyset]$.

Let us return to the example depicted in Fig. 1(a). Place $p_4$ is concurrent to signals $a$ and $c$. Therefore, $\mathcal{C}_{p_4} = (-0-1)$. Place $p_4$ belongs to QPS($d+$), but if $\mathcal{C}_{p_4}$ takes part in the approximation of $\widehat{\text{GQR}}(d+)$, the quiescent region will be erroneously overestimated due to the covering of the code $(0001) \in \widehat{\text{GER}}(d-)$. This fact can be observed by the existence of a structural conflict between places $p_4$ and $p_{14}$: $\mathcal{C}_{p_4} \cdot \mathcal{C}_{p_{14}} = (-0-1) \cdot (0001) = (0001)$.

The absence of *structural coding conflicts* guarantees the correctness—although conservatively—of the structural approximations of the ER's and QR's. Property 12 states that under this condition, quiescent regions are properly approximated and the cover subtractions required to avoid the overestimation of the QR's are safe. Property 13 guarantees the proper approximation of excitation regions. It has also been proved that for free-choice STG's, the absence of structural coding conflicts guarantees the absence of USC conflicts [30].

*Property 12 (Correct QR Approximation):* Given an SM-cover in which the STG is free of coding conflicts, then for any transition $a_{i^*}$, 1) no binary code of reachable markings outside QR($a_{i^*}$) is covered by the cover function of any place in QPS($a_{i^*}$) and 2) each marking in QR($a_{i^*}$) is covered by the cover function of some place in QPS($a_{i^*}$).

*Proof:*

1) Let us assume that there exists a marking $M \notin \text{QR}(a_{i^*})$ and a place $p_1 \in \text{QPS}(a_{i^*})$ such that $\lambda(M) \sqsubseteq \mathcal{C}_{p_1}$. Take an SM-component in the SM-cover that contains $p_1$. Since $M$ is not in QR($a_{i^*}$), then $M$ should be covered by the cover cube of some other place $p_2 \notin \text{QPS}(a_{i^*})$ of the SM-component. Therefore, we are contradicting the absence of structural coding conflicts.
2) Every marking $M \in \text{QR}(a_{i^*})$ marks some place that is interleaved with a pair $(a_{i^*}, a_{j^*})$, $a_{j^*} \in \text{next}(a_{i^*})$. All these places are included in QPS($a_{i^*}$), and for all of them [except $p \in {}^\bullet(a_{j^*})$], the cover cube $\mathcal{C}_p$ is directly used as a cover function. The cover cube $\mathcal{C}_p$ is conservative; that is, it covers all the markings corresponding to the marked region of $p$, and probably some vertices in the dc-*set* or other markings.

The marked region can only be underestimated by the cover functions for places $p \in {}^\bullet(a_{j^*})$ because of their recomputation into $(cv(p) = cv(p) - cv(a_{j^*}))$. If for some reason the cover function $cv(a_{j^*})$ is overestimated, we can incorrectly eliminate some binary codes from $cv(p)$. Suppose a marking $M \in \text{MR}(p)$ and its binary code $\lambda(M)$ is covered by both $cv(p)$ and $cv(a_{j^*})$ $(\lambda(M) \sqsubset cv(p) \cdot cv(a_{j^*}))$. Let us also assume that marking $M$ is not included in ER($a_{j^*}$). This is a case of ER($a_{j^*}$) overestimation, in which the recomputation of $cv(p)$ will lead to an underestimation of MR($p$) because the binary code $\lambda(M)$ will be incorrectly eliminated from the cover. Now we will show that this potential situation contradicts the assumption that structural coding conflicts exist.

Due to the assumption that $M \notin \text{ER}(a_{j^*})$, some place $p_1 \in {}^\bullet(a_{j^*})$ has to be unmarked in $M$ [otherwise $M \in \text{ER}(a_{j^*})$]. Take one SM from the SM-cover that includes this place $p_1$. Because of the liveness of the STG (Property 7), another place $p_2$ that is marked at $M$ $(M \in \text{MR}(p_2))$ should also exist in SM. Then, the intersection of the cover cubes for both places $p_1$ and $p_2$ in SM is nonempty $(\mathcal{C}_{p_1} \cap \mathcal{C}_{p_2} \neq \emptyset)$ because $\lambda(M) \sqsubset \mathcal{C}_{p_2}$ and $\lambda(M) \sqsubset \mathcal{C}_{a_{j^*}} \sqsubseteq \mathcal{C}_{p_1}$. Therefore, the condition on the absence of structural coding conflicts in the STG is violated. ■

*Property 13 (Correct ER Approximation):* Given an SM-cover in which the STG is free of coding conflicts, for any transition $a_{i^*}$, no binary code of reachable markings outside ER($a_{i^*}$) is covered by the cover function $cv(a_{i^*})$.

*Proof:* The proof can be carried out similarly to Property 12. Basically, it is necessary to take a place $p$ in the preset of $a_{j^*}$ and a SM-component in the SM-cover that contains $p$. Then, due to Property 7, any marking outside ER($a_{j^*}$) covered by $cv(a_{j^*})$ will be detected as a structural coding conflict between $p$ and some other place $p'$ in the selected SM-component. ■

As an example, take the STG depicted in Fig. 1(a) and its SM-cover in Fig. 2. Several structural coding conflicts can be detected in this STG. For $\text{SM}_1$, $\mathcal{C}_{p_1} \cdot \mathcal{C}_{p_2} \neq \emptyset$, $\mathcal{C}_{p_{10}} \cdot \mathcal{C}_{p_{11}} \neq \emptyset$, and $\mathcal{C}_{p_4} \cdot \mathcal{C}_{p_{14}} \neq \emptyset$. For $\text{SM}_2$, $\mathcal{C}_{p_{10}} \cdot \mathcal{C}_{p_{11}} \neq \emptyset$. Last, for $\text{SM}_3$; $\mathcal{C}_{p_8} \cdot \mathcal{C}_{p_{12}} \neq \emptyset$, and $\mathcal{C}_{p_6} \cdot \mathcal{C}_{p_{10}} \neq \emptyset$. Therefore, not enough information is contained in the cover cubes in order to precisely approximate the signal regions.

From Properties 12 and 13, it follows that if there exists an SM-cover under which an STG is free from structural coding conflicts, the approximations of QR's and ER's are safe and can be used for synthesis. If an STG has structural coding conflicts, we should go for the refinement of the approximations. Note that the presence of structural coding conflicts in STG does not necessarily lead to the violation of the correctness conditions for the covers. The conflicts may be due to intersections on a vertex in the dc-*set*, or the original STG satisfied the CSC condition instead of the USC (two markings that have the same binary code but that are valid for the synthesis process). This later possibility increases the complexity of the analysis and will be addressed in Section VII. Thus, we have two possibilities in the synthesis process.

1) To *refine* the cover functions. The refinement technique leads to a growth of the number of cubes in the cover but provides more accurate approximations.
2) When no successful refinements can be applied, to be *conservative* (because still the intersection may be at the dc-*set*) and consider each cube intersection as a real structural coding conflict. Then, by adding state signals, the covers can always be reduced to nonintersecting.

The latter approach was presented in [27]. In the following, we will concentrate only on the refinement techniques in the synthesis process.

## VII. Refinement of Cover Functions

The previous section has shown that the structural approximations provided for the signal regions may be either overestimated or underestimated. The lack of accuracy can be checked by the existence of structural coding conflicts. Therefore, before going into the SI synthesis process, the accuracy of the approximations has to be increased.

This section presents a refinement mechanism that increases the accuracy of the cover functions. The refinement process is carried out by taking additional information from the SM-components of the STG. At first, we provide a general view on the refinement process that uses SM's, while the rest of the section discusses how to check when the refinement process is needed, which SM's should be used for each refinement, and

under which conditions the refinement process guarantees the desired accuracy of the approximations.

The generation of the structural approximations for ER's and QR's must combine both the creation of the cover functions for places and transitions and its refinement. This process is carried out in four steps.

1) The domain of the approximations and the initial cover functions for places are computed.
2) The cover functions are refined in case structural coding conflicts exist.
3) Cover functions for transitions are constructed by using covers for places.
4) Cover functions for places at the boundaries of the QR regions are recomputed by subtracting the necessary cover functions of transitions.

### A. General View of Refinement Process

If two overestimated cover functions corresponding to the different implied values of the same signal [e.g., one function for QR$(a+)$ while the other for ER$(a-)$] have nonempty intersection the synthesis process cannot be carried out. These functions should be refined. The presence of structural coding conflicts is the condition by which one can check whether refinement is necessary. If there are no structural coding conflicts in the STG, then there is no need for the refinement of cover functions.

Note that the initial approximation for cover cubes of places or transitions can be rough as they are based on the concurrency relations between nodes and signals. This relation is not sufficient for an accurate characterization of the dynamic behavior of the STG, e.g., from a transition $a_{i^*}$ concurrent to $b_{j^*}$ and $a_{i^*}$ concurrent to $c_{k^*}$, nothing can be said about the joint concurrency of $a_{i^*}$, $b_{j^*}$, and $c_{k^*}$ (in fact, transitions $b_{j^*}$ and $c_{k^*}$ could be ordered). To exploit the structure of causal relations between STG nodes more exactly, we should refine the initial approximation for place or transition cover cubes.

The idea of refinement is based on the observation that each SM-component presents a partial behavior of STG, while the composition of all SM-components from an SM-cover gives a complete behavior. Therefore, if we take a cover cube for some place in one SM-component and intersect this cube with the cubes for all places of another SM-component, we will get the refined cover function of the place because intersection of the cubes in Boolean domain corresponds to the composition operation in STG domain. Actually, it is sufficient to perform the intersection only between places of SM-components that are mutually concurrent because the intersection of marked regions of nonconcurrent places is empty.

Formally, the *refinement* of the cover function $cv(p)$ by an SM-component SM results in the cover that is obtained after restricting $cv(p)$ to the sum of the cover cubes of any place $p_i \in$ SM that is concurrent to $p$; that is

$$cv(p) = \sum_{p_i \in \mathrm{SM}:\,(p,p_i)\in\mathcal{CR}} \mathcal{C}_{p_i} \cdot cv(p).$$

The refinement algorithm for a place $p_1$ with respect to the SM-component SM is described in Fig. 11. Property 7 guarantees that the refinement procedure is safe; that is, no marking in the marked region of a place can be left uncovered after applying a refinement because all reachable markings are covered by some $cv(p)$ in the SM's used for refinement. The cover function for transition can be indirectly refined by recomputing the intersection of its predecessor places, $cv(a_{i^*}) = \bigcap_{p\in{}^\bullet(a_{i^*})} cv(p)$. Clearly, a large number of refinements increases the support of the cover functions, improving the accuracy of the approximations. However, such an approach has two shortcomings.

```
refine_cv (p1, SM, CR) {
 cv = ∅;
 foreach (place p2 ∈ SM : (p1, p2) ∈ CR) do cv = cv + cv(p2);
 return cv · cv(p1);
}
```

Fig. 11. Cover function refinement algorithm.

1) It increases the number of cubes to be processed that in the extreme case it may be comparable to the number of markings.
2) The question about the minimal set of SM-components that is sufficient to avoid all overestimations is still an open problem. Even though the number of SM's that can be generated for a PN is potentially exponential, the sufficiency of the refinement process has not been guaranteed.

### B. Elimination of Cover Overestimations

This subsection presents the details of the refinement technique based on the utilization of an SM-cover. At first, the conditions under which an SM can be used to apply a successful refinement are described. Even though these conditions are not sufficient for the removal of all structural coding conflicts, their application is quite efficient in practice. Finally, we show that the absence of structural conflicts guarantees that STG satisfies the CSC requirement.

*1) Cover Function Refinement:* This section shows how a structural coding conflict detected in one SM-component can be eliminated by refinement of a cover function by another SM component.

Let us assume that an STG contains an SM-component $\mathrm{SM}_i \in$ SM-cover, such that $\mathrm{SM}_i$ has two places $p_1$ and $p_2$, $\mathcal{C}_{p_1} \cdot \mathcal{C}_{p_2} \neq \emptyset$. From Definition 11, we infer that there might exist markings $M_1$ and $M_2$ such that $M_1 \in \mathrm{MR}(p_1)$, $M_2 \in \mathrm{MR}(p_2)$, and $\lambda(M_1) = \lambda(M_2)$, implying a structural coding conflict between places $p_1$ and $p_2$. However, markings $M_1$ and $M_2$ may be unreachable, and the structural coding conflict might actually occur due to the overestimation of the marked regions by their cover functions. Nevertheless, being conservative, we must assume that $M_1$ and $M_2$ are reachable unless it can be disproved.

The information provided by another SM-component in the SM-cover may help to eliminate the overestimation. Suppose we could find an SM-component $\mathrm{SM}_j$ that contains $p_1$ but does not contain $p_2$. If both $M_1$ and $M_2$ are reachable markings, then $M_1$ belongs to $\mathrm{MR}(p_1)$ in $\mathrm{SM}_j$, while for $M_2$ there exists a place $p_3$ such that $M_2 \in \mathrm{MR}(p_3)$. Therefore, $\mathcal{C}_{p_1} \cdot$

```
cod_refine_cv (STG,SMC,CR){
 foreach (place p ∈ STG) do
   if (∃SM1 SM2 ∈ SMC : p ∈ SM1, SM2 ∧ p has struc. conf. in SM1
      ∧ p has no structural conflict in SM2) then
      foreach (p' ∈ STG) do cv(p'):=refine_cv(p', SM2, CR);
}
```

Fig. 12. Cover function marking coding refinement algorithm.

$\mathcal{C}_{p_3} \neq \emptyset$; which means that place $p_1$ should have a structural coding conflict in every SM-component (see Property 7). (The motivation for this fact is that any reachable marking should be included in some marked region.)

Conversely, if $\mathrm{SM}_j$ contains place $p_1$ but does not contain any other place $p_3$ for which $\mathcal{C}_{p_1} \cdot \mathcal{C}_{p_3} \neq \emptyset$, then we can conclude that $M_2$ is not a reachable marking, and the structural coding conflict between $p_1$ and $p_2$ is *fake* (happens only due to an overestimation of $p_2$) [27], [30]. Additionally, it can be guaranteed that the SM-component $\mathrm{SM}_j$ can be used to effectively *refine* the cover function of place $p_2$ and eliminate the overestimation. Since no place $p_3 \in \mathrm{SM}_j$ has a structural conflict with $p_2$, no cover cube $\mathcal{C}_{p_3}$ in $\mathrm{SM}_j$ covers $\lambda(M_1)$.

The refinement for $p$ is computed $cv(p) = refine\_cv(p, \mathrm{SM}_j, \mathcal{CR})$, and after the refinement, the cover function $cv$ does not have structural coding conflicts in $\mathrm{SM}_i$. The procedure depicted in Fig. 12 refines the cover functions of places in the STG when fake structural conflicts are detected. Note that refinements concern not only the place with structural conflicts but all the places in the STG. This is done because we found that in practice, places closer to other places with fake structural conflicts have also overestimated cover functions. Even though the overestimation could be in the dc-*set*, our experiments show that this more general application of refinement leads to much better minimization solutions.

The example in Fig. 1(a) contains three structural coding conflicts at $\mathrm{SM}_1$ (Fig. 2)

$$(p_1, p_2)\text{:}\mathcal{C}_{p_1} \cdot \mathcal{C}_{p_2} = (0000) \cdot (-0-0) = (0000)$$
$$(p_4, p_{14})\text{:}\mathcal{C}_{p_4} \cdot \mathcal{C}_{p_{14}} = (-0-1) \cdot (0001) = (0001)$$
$$(p_{10}, p_{11})\text{:}\mathcal{C}_{p_{10}} \cdot \mathcal{C}_{p_{11}} = (1111) \cdot (-111) = (1111).$$

Places $p_2$ and $p_4$ do not have structural coding conflicts at $\mathrm{SM}_2$. Therefore, this SM-component can be used to refine the corresponding cover functions

$$cv(p_2) = cv(p_2) \cdot (\mathcal{C}_{p_3} + \mathcal{C}_{p_5}) = ((1000) + (-010))$$
$$cv(p_4) = cv(p_4) \cdot (\mathcal{C}_{p_3} + \mathcal{C}_{p_5}) = ((1001) + (-011)).$$

The technique for the resolving the structural conflict between $p_{10}$ and $p_{11}$ is different and is discussed further.

*2) Refinement Technique and CSC Property:* Refinement does not work if the structural coding conflict for places $p_1$ and $p_2$ (in Fig. 1) corresponds to reachable markings $M1$ and $M2$ ($M1 \in \mathrm{MR}(p_1)$ $M2 \in \mathrm{MR}(p_2)$). However, the correctness of the cover [see (2)] is not violated if a coding conflict corresponds to markings $M1$ and $M2$ that satisfy the CSC property. The structure of the STG provides a sufficient condition to find whether the structural coding conflict satisfies the CSC property.

*Theorem 14 (Sufficient Condition for CSC):* If an STG has a CSC violation, then in a given SM-cover SMC, one can find an SM-component SM containing a pair of places $p_i$ and $p_j$ such that:

1) $p_i$ is in the preset of an output transition $a_{i^*}$;
2) $p_j$ is not in the preset of any other transition of signal $a$;
3) $\widehat{\mathrm{ER}}(a_{i^*}) \cdot \widehat{\mathrm{MR}}(p_j) \neq \emptyset$.

*Proof:* A CSC violation $(M_1, M_2)$ means that there exists an output signal $a$ such that $M_1 \in \mathrm{ER}(a_{i^*})$ and $\nexists a_{j^*}$: $M_2 \in \mathrm{ER}(a_{j^*})$. Let us assume that $a_{j^*}$ is the first transition of signal $a$ that can be enabled in a feasible sequence starting from $M_2$, i.e., $\exists\sigma$: $M_2[\sigma a_{j^*}\rangle$ and no other transition of signal $a$ is enabled in $\sigma$. Since $M_2 \notin \mathrm{ER}(a_{j^*})$, there is at least one place $p' \in {}^\bullet(a_{j^*})$ that is not marked in $M_2$. Let us take an SM-component SM including place $p'$. According to the STG liveness, $M_2$ and $M_1$ should hold a token in places $p_j$ and $p_i$, respectively ($p_j, p_i \in \mathrm{SM}$, where $p_i$ is an input place to $a_{j^*}$). Clearly, by the choice of $\sigma$, place $p_j$ cannot be in a preset of any transition $a_{j^*}$, and Condition 2) of the theorem is satisfied. Taking into account that $M_1 \in \mathrm{ER}(a_{i^*}) \subseteq \mathrm{ER}(p_i)$ and $M_2 \in \mathrm{MR}(p_j)$, we can conclude that $\widehat{\mathrm{ER}}(a_{i^*}) \cdot \widehat{\mathrm{MR}}(p_j) \neq \emptyset$. ■

*Theorem 15 (Detection of Fake Coding Conflicts):* An STG satisfies the CSC property if for any place $p_j$ in the preset of an output signal transition $a_{i^*}$, there exists an SM-component SM in the SMC including place $p_j$ such that SM does not contain any structural coding conflict for $p_j$; i.e., $\forall p_j \in {}^\bullet(a_{i^*})$, $a \in A_O$: $\exists \mathrm{SM} \in \mathrm{SMC}$, $p_j \in \mathrm{SM}$: $\forall p_k \in \mathrm{SM}$, $j \neq k$, $\Rightarrow \widehat{\mathrm{MR}}(p_j) \cdot \widehat{\mathrm{MR}}(p_k) = \emptyset$.

*Proof:* Let us assume the existence of a CSC violation due to markings $M_1$ and $M_2$. From Theorem 14, there should exist an SM-component $\mathrm{SM}_1 \in \mathrm{SMC}$ containing two places $p_i$, $p_j$, and a transition $a_{i^*}$ such that $p_i \in {}^\bullet(a_{i^*})$ and $M_1 \in \mathrm{ER}(a_{i^*})$, $M_2 \in \mathrm{MR}(p_j)$, but $M_2 \notin \mathrm{MR}(p_i)$.

We will prove that if there exists $\mathrm{SM}_2 \in \mathrm{SMC}$ that contain both nodes $p_i$ and $a_{i^*}$ without coding conflicts for place $p_i$, then the assumed CSC violation is contradicted. Since $M_2 \notin \mathrm{MR}(p_i)$, there should exist a place $p_k \in \mathrm{SM}_2$ such that $M_2 \in \mathrm{MR}(p_k)$. Hence, a coding conflict should exist between places $p_i$ and $p_k$. But place $p_i$ does not contain any coding conflict in $\mathrm{SM}_2$, which contradicts the assumption about the CSC violation. ■

Both Theorems 14 and 15 provide the conditions to eliminate structural coding conflicts in specifications that satisfy the CSC condition.

Let us go back to the structural coding conflict between places $(p_{10}, p_{11})$ at $\mathrm{SM}_1$ of the STG in Fig. 1(a). This coding conflict cannot be eliminated by means of refinement because place $p_{10}$ has the same coding conflict at $\mathrm{SM}_2$, and a coding conflict $(p_{10}, p_6)$ at $\mathrm{SM}_3$. However, the conflict between places $p_{10}$ and $p_6$ satisfies Theorem 14. Note that $a-/1 \in (p_6)^\bullet$ and $a-/2 \in (p_{10})^\bullet$; therefore, if it would correspond to a real CSC conflict, there would exist some other place not in the preset of any transition of signal $a$ holding a conflict with place $p_6$. Since that is not the case, this conflict can be related to markings that satisfy the CSC condition. Last, it can be concluded that place $p_{10}$ has no conflicts and $\mathrm{SM}_3$ can

be used to determine that the conflict between $p_{10}$ and $p_{11}$ is fake at both $SM_1$ and $SM_2$.

## VIII. Synthesis Methodology

This section completes the synthesis process by applying the signal region approximations to the design of an SI-circuit under a particular architecture. For simplicity, we have selected the *atomic complex gate per excitation function* architecture. This work proposes a two-step heuristic synthesis algorithm. Initially, nonoptimized set and reset excitation functions that satisfy the implementability conditions (correctness and monotonicity) are derived. Starting from these covers, several minimizations are applied to simplify the functions while maintaining the implementability conditions. However, every time minimization is applied, the algorithm must determine whether the final result is speed independence or not. Therefore, both correctness and monotonicity should be structurally verified before accepting the minimization.

### A. Initial Excitation Functions

The set and reset functions for a signal $a$ must cover all binary codes in its rising and falling generalized excitation regions. Since markings in GER's are obtained by combining the particular ER's, set and reset covers can be computed as the union of covers for transitions, i.e., $S(a) = \bigcup_{\forall a_{i+}} C(a_{i+})$ and $R(a) = \bigcup_{\forall a_{i-}} C(a_{i-})$. Property 13 guarantees that under the absence of structural conflicts, the cover functions are correct covers for $\widehat{\mathrm{ER}}(a_{i^*})$; therefore, $C(a_{i+}) = cv(a_{i+})$ and $C(a_{i-}) = cv(a_{i-})$. Following Theorem 15, a cover cube $\mathcal{C}(a_{i^*})$ does not overestimate $\widehat{\mathrm{ER}}(a_{i^*})$ if no predecessor place $p \in {}^{\bullet}(a_{i^*})$ needs refinement. Structural coding conflicts are checked in the SM-cover. If they exist, refining or inserting state signals is necessary. The absence of structural coding conflicts guarantees the CSC property [27] and the existence of correct covers.

By applying this scheme to signal $d$ in Fig. 1, we obtain $S(d) = cv(d+/1) + cv(d+/2)$. As place $p_2$, corresponding to $cv(d+/1)$, is involved in a structural conflict, its cover cube $\mathcal{C}_{p_2} = (-0-0)$ is refined into a set of binary codes $\{1000, 1010, 0010\}$. Place $p_9$, corresponding to $cv(d+/2)$, is free of structural conflicts, and its cover cube does not need any refinement. As a result, $S(d) = \{1000, 1010, 0010\} + \{1110\}$. Similarly, we can obtain $R(d) = \mathcal{C}_{p_{14}} = \{0001\}$.

### B. Checking the Synthesis Conditions

From the initial set of covers, multiple minimization techniques will be tried in order to simplify the final implementation. Some of these transformations can be directly applied without further correctness or monotonicity checking because they are known to preserve these properties. However, any minimization technique that implies increasing the number of markings covered by the set/reset covers requires checking the SI synthesis conditions to guarantee the speed independence of the result.

*1) Correctness:* The correctness condition [see (2)] requires all binary codes of markings in $\mathrm{GER}(a+)$ ($\mathrm{GER}(a-)$) to be covered by $S(a)$ ($R(a)$). This condition defines the on-set ($\mathrm{on}(f_a)$) of the function and can be verified as: $\forall a_{i+}$: $cv(a_{i+}) \subseteq S(a)$, and $\forall a_{i-}$: $cv(a_{i-}) \subseteq R(a)$.

Also, no binary code of markings inside $\mathrm{GER}(a-) \cup \mathrm{GQR}(a-)$ ($\mathrm{GER}(a+) \cup \mathrm{GQR}(a+)$) can be used in the minimization of $S(a)$ ($R(a)$). This condition defines the off-set ($\mathrm{off}(f_a)$) of the function, and can be verified as: $\forall a_{i-}$: $[S(a) \cdot cv(a_{i-}) = \emptyset \wedge \forall p \in \mathrm{QPS}(a_{i-})$: $S(a) \cdot cv(p) = \emptyset]$, and $\forall a_{i+}$: $[R(a) \cdot cv(a_{i+}) = \emptyset \wedge \forall p \in \mathrm{QPS}(a_{i+})$: $R(a) \cdot cv(p) = \emptyset]$.

*2) Monotonicity:* The *monotonicity* condition has to be checked for each cover by using a two-step technique. To simplify the reasoning, let us assume that $C(a_{i^*})$ contains exactly one cube.

Assuming the correctness of the cover $C(a_{i^*})$, it implies that the cube will be turned on at $\mathrm{ER}(a_{i^*})$ but should be turned off somewhere inside $\mathrm{QR}(a_{i^*})$ or before reaching the following ER's. Then, it cannot be turned on again inside the quiescent region without violating the monotonicity condition; that is, the cover $C(a_{i^*})$ can only be switched on to implement transition $a_{i^*}$ (see Definition 1).

The monotonicity condition can be structurally verified by determining the border places in $\mathrm{QPS}(a_{i^*})$ in which the cover cube still can be ON, while in their successors it should be turned OFF.

Let us define $C(a_{i^*}) \downarrow$ as the set of transitions in $\mathcal{IR}(a_{i^*}, a_{j^*})$ [where $a_{j^*} \in \mathrm{next}(a_{i^*})$] that will turn off $C(a_{i^*})$ for the first time. Let us also generalize the *interleaving relation* for the pairs $(a_{i^*}, t)$ and $(t, a_{j^*})$, where $t \in S(a_{i^*}) \downarrow$. To guarantee the monotonicity condition, given any place $p$ in $\mathrm{QPS}(a_{i^*})$ that is interleaved in $\mathcal{IR}(t, a_{j^*})$ ($p$ is reached *after* $t$), the intersection between the cover $C(a_{i^*})$ and the cover function $cv(p)$ should be empty. This is characterized formally in the following property.

*Property 16 (Structural Checking of Monotonicity):* The correct cover $C(a_{i^*})$ is monotonic if for any $a_{j^*} \in \mathrm{next}(a_{i^*})$ any $t \in S(a_{i^*}) \downarrow$ and any place $p \in \mathcal{IR}(t, a_{j^*})$, the cover $C(a_{i^*})$ does not intersect with $cv(p)$.

*Proof:* If a cover $C(a_i*)$ is correct (2), then $C(\mathcal{I}^i_{a^*})$ has to be turned off somewhere inside $\mathrm{QPS}(a_{i^*})$. By examining the transitions that are in $\mathcal{IR}(a_{i^*}, a_{j^*})$ [where $a_{j^*} \in \mathrm{next}(a_{i^*})$], we can find the set of transitions $C(a_{i^*}) \downarrow$ turning off $C(a_{i^*})$ for the first time. Note that none of the literals corresponding to transitions before reaching $C(a_{i^*}) \downarrow$ can be present in the cube $C(a_{i^*})$.

To be monotonic, once $C(a_{i^*})$ is turned off by a transition in $C(a_{i^*}) \downarrow$, the cube cannot be turned on again inside $\mathrm{QPS}(a_{i^*})$.

The marked region of all sequences of places that are in $\mathcal{IR}(a_{i^*}, t)$ is covered by $C(a_{i^*})$. Conversely, all places that are in $\mathcal{IR}(t, a_{j^*})$ can be reached only after the firing of $t$; that is, after the cube $C(a_{i^*})$ is turned off. Therefore, monotonicity is ensured if cube $C(a_{i^*})$ is never turned on again in the markings that are covered by the marked regions of places $p \in \mathcal{IR}(t, a_{j^*})$. ■

As an example, let us assume that we have computed the cover $C(d-) = \{000-\}$ for the STG in Fig. 1. The set $C(d-) \downarrow$ will contain transitions $\{a+/1, b+/2\}$; therefore, $C(d-)$ is monotonic because it can intersect with the covers

for place $p_1$ but cannot intersect with the covers of any place interleaved between $a+/1 \rightarrow d+/1$ ($p_2$) and $b+/2 \rightarrow d+/2$ ($p_7$, $p_8$, $p_9$).

When $C(a_{i^*})$ has several cubes, the monotonic sequences defined by the set $C(a_{i^*}) \downarrow$ are conservatively computed. A transition belongs to the set $C(a_{i^*}) \downarrow$ if it is the first one such that the cover cubes of the places in its postset are not completely covered by $C(a_{i^*})$, i.e., $\exists p \in (t)^\bullet$: $p \in \text{QPS}(a_{i^*}) \wedge cv(p) \not\sqsubseteq C(a_{i^*})$.

### C. Synthesis Algorithm

From the initial set of covers, several minimizations are heuristically applied. (A detailed description of each minimization is described in the Appendix.) For simplicity, we assume that the STG satisfies the CSC condition; otherwise, state encoding techniques are applied [30]. Additionally, safeness, liveness, and consistency on the STG should be checked beforehand [12], [31]. The selected minimization process is the following.

1) Each set/reset cover is expanded toward the quiescent regions and dc-*set* by eliminating literals.
2) After expansion toward the quiescent region, covers are checked to be complete; that is, if the set (reset) cover includes all binary codes in $\widehat{\text{GQR}}(a+)$ ($\widehat{\text{GQR}}(a-)$), then the atomic complex gate per signal architecture can be used, hence avoiding the use of a C-latch.
3) Signals that cannot be directly implemented by the set or reset cover, i.e., requiring the memory element, can be further expanded toward the quiescent region of its predecessor transitions (see the Appendix).
4) The C-latch can be collapsed with the set and reset covers, leading to a potential simplification of the circuit.
5) The overall synthesis process is completed by creating the circuit and mapping its different elements onto a gate library.

To demonstrate the evolution of the covers through the minimization process, the synthesis algorithm will be applied to the output signal $d$ in Fig. 1. The previously computed initial covers are $C(d+/1) = \{10-0, -010\}$, $C(d+/2) = \{1110\}$, $C(d-) = \{0001\}$, and in the first step of the minimization process are expanded toward the quiescent region and dc-*set*.

Literal $d$ can be eliminated from the support of $C(d+/1)$ including markings $\{s_3, s_5, s_8\}$ in the cover, which results in $C(d+/1) = \{10--, -01-\}$. Literal $b$ can be eliminated from both $(10--)$ and $(11--)$, generating the cover $C(d+/1) = \{1---, -01-\}$. When simplifying the cover $C(d+/2) = \{1110\}$, literal $c$ can be eliminated, expanding the cover toward the dc-*set*, which results in $C(d+/2) = \{11-0\}$. Last, literal $d$ can be eliminated from $(11-0)$, obtaining $C(d+/2) = \{11--\}$. Both covers are used to implement the set function $S(d) = \{1---, -01-\}$. With respect to $C(d-)$, literal $d$ can be eliminated by expanding the cover toward the quiescent region and obtaining $R(d-) = C(d-) = \{000-\}$.

For this particular signal, complete cover minimization nor backward expansion nor memory collapsing can be applied. The final implementation is depicted in Fig. 4(b).

## IX. Experimental Results

This section presents a number of experiments that evaluate the quality of the proposed synthesis methodology. Four relevant issues have been analyzed: 1) the influence of minimization on the final area of circuits, 2) area results compared to previous synthesis methodologies, 3) CPU speedup due to the structural algorithm compared to state-based algorithms, and 4) the relation among markings in the STG's, the number of cubes required for the structural approximations, and the quality of area minimizations. Note that all synthesis results have been formally verified to be speed independent [32]. The CPU times have been obtained on a Sun SPARC20 workstation.

In all tables, columns labeled $P$, $T$, and $RG$ indicate the number of places, transitions, and reachable markings. Columns labeled $C$ and SM denote the number of cubes and SM's required by structural algorithms. These values give an intuitive idea about the complexity of each benchmark.

### A. Heuristics for Area Minimization

This section compares the average area improvement obtained in two benchmark sets (see Fig. 13). In both cases, the process starts from an initial semioptimized implementation, in which only expansions toward the quiescent region and dc-*set* have been applied, and progressively evolves toward more efficient implementations.

Points in the column labeled **raw** are the initial semioptimized implementation. Progressively, in column **me**, transitions are allowed to be merged; in **co**, *complete* signal networks are detected. Memory element collapsing is applied at **si**. Last, region covers are expanded toward the backward quiescent regions in **ba** (see the Appendix). From a technology-independent implementation, a Boolean-matching mapping algorithm is applied [33]. The column labeled **map** presents the results obtained after the application of a technology-mapping step that, for example, merges simple gates into complex ones when available in the library (currently complex gates up to four inputs such as AOI22).

### B. Area of the Circuits

Table V compares the area results of several synthesis tools including our methodology. The goal of this experiment is to show that even though structural techniques only approximate the reachable markings in the STG's, this methodology does not negatively influence the quality of the circuits.

Columns labeled SYN and FCG report the area obtained by the synthesis methodologies developed at Stanford [24] and Aizu [19]. Columns labeled S3C contain area results for our methodology without using the backward minimization and mapping (left column) and fully minimized (right column).

The results show that the new logic-minimization techniques provide significant improvements—23% area reduction with respect to [24]—in short CPU times—less than 8 s for the worst case (pe-send-ifc). We also took into account that some of the new minimization techniques were not fully

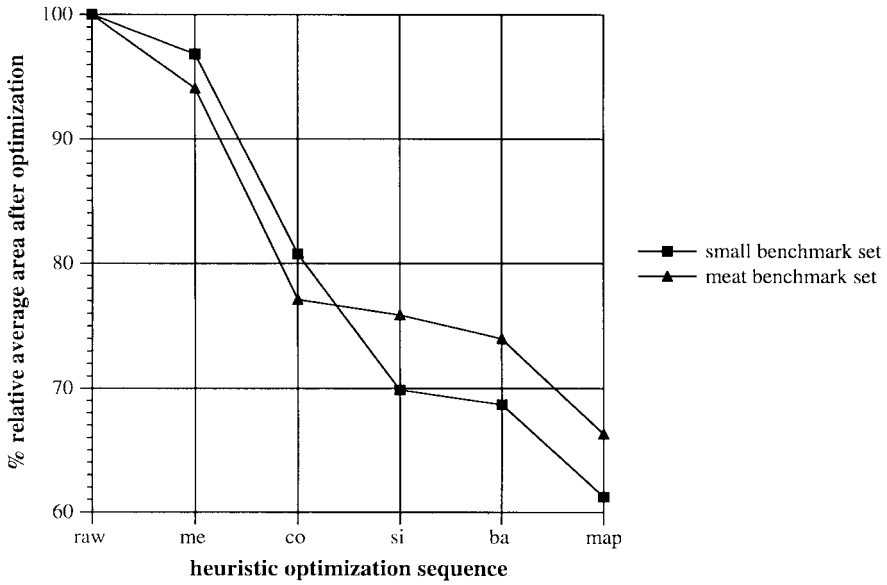



Fig. 13. Average minimization results for the benchmark sets.

TABLE V
AREA RESULTS COMPARISON WITH TOTALS BY SYN (1) AND BY FORCAGE (2) (* NONFREE-CHOICE—NONAVAILABLE RESULT)

| STG | RG | C | SYN | FCG | S3C | | STG | RG | C | SYN | FCG | S3C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| chu133 | 24 | 50 | 232 | 216 | 226 | 208 | wrdatab | 216 | 110 | 784 | 744 | 680 | 488 |
| chu150 | 26 | 58 | 248 | 232 | 152 | 128 | xyz | 8 | 20 | 200 | 192 | 168 | 136 |
| chu172 | 12 | 25 | 168 | 112 | 104 | 104 | alloc-outbound | 21 | 43 | 400 | - | 348 | 308 |
| converta | 18 | 36 | 376 | 320 | 400 | 258 | mp-forward-pkt | 22 | 62 | 320 | - | 320 | 256 |
| ebergen | 18 | 36 | 352 | 280 | 176 | 120 | nak-pa | 58 | 62 | 336 | - | 320 | 320 |
| full | 16 | 28 | 112 | 112 | 80 | 80 | pe-rcv-ifc(*) | 65 | 147 | 1304 | - | 1296 | 1146 |
| hazard | 12 | 26 | 248 | 240 | 80 | 80 | pe-send-ifc(*) | 117 | 228 | 1632 | 1632 | 1258 | 1122 |
| hybridf | 80 | 62 | 152 | 152 | 130 | 130 | ram-read-sbuf | 39 | 82 | 432 | - | 400 | 360 |
| nowick | 20 | 47 | 456 | 456 | 322 | 274 | rcv-setup | 14 | 31 | 144 | - | 152 | 120 |
| qr42 | 18 | 36 | 352 | 280 | 176 | 120 | sbuf-ram-write | 64 | 77 | 320 | - | 328 | 304 |
| rpdft | 22 | 44 | 224 | - | 200 | 160 | sbuf-read-ctl | 19 | 41 | 296 | - | 298 | 258 |
| trimos-send | 336 | 129 | 648 | - | 672 | 552 | sbuf-send-ctl | 27 | 59 | 280 | - | 322 | 226 |
| vbe10b | 256 | 98 | 792 | 784 | 608 | 608 | sbuf-send-pkt2 | 28 | 59 | 504 | - | 426 | 364 |
| vbe5b | 24 | 58 | 192 | 192 | 240 | 208 | sendr-done | 9 | 19 | 80 | - | 82 | 82 |
| vbe5c | 24 | 38 | 200 | 200 | 152 | 152 | | | | | | | |
| **Total**(1) | | | | | | | | | | 11784 | - | 10016 | 9076 |
| **Total**(2) | | | | | | | | | | 6496 | 6144 | 5032 | 4216 |

used by SYN and FORCAGE (e.g., backward expansions and mapping). Thus, for the sake of comparison fairness, we disabled such optimizations, still obtaining a 15% improvement. Therefore, we can conclude from the experimental results that structural methods, even being conservative, do not influence negatively on the quality of the final result.

### C. CPU Time: Structural Versus State Based

To illustrate the effectiveness of structural over state-graph-based methods, we have run some experiments for STG's with a large reachability graph, comparing CPU times with SIS [6] and ASSASSIN [8] (see Table VI). The superiority of structural methods is evident.

Table VII reports the CPU times for two large scalable benchmarks. The *dining philosophers* benchmark is one of the examples that illustrates that nonfree-choice STG's can also be synthesized if a cover of state machines can be found for the net. Another scalable example is the *Muller pipeline*. Its STG contains no choice places, and the circuit realization is a chain of C-latches.

TABLE VI
CPU TIME FOR SYNTHESIS: COMPARISON WITH SIS AND ASSASSIN

| STG | P | T | RG | C | SM | area | CPU SIS | CPU ASS. | CPU S3C |
|---|---|---|---|---|---|---|---|---|---|
| chu133 | 17 | 14 | 24 | 50 | 4 | 208 | 0.1 | 0.4 | 0.3 |
| vbe10b | 32 | 22 | 256 | 98 | 6 | 608 | 4 | 12 | 1 |
| trimos-send | 30 | 18 | 336 | 129 | 7 | 552 | 6 | 19 | 1 |
| tsbmsi | 40 | 28 | 1024 | 143 | 10 | 744 | 43 | 256 | 1 |
| tsbmSIBRK | 68 | 50 | 4730 | 298 | 15 | 1136 | 1876 | 12219 | 8 |
| mread-1 | 40 | 28 | 2254 | 117 | 8 | 610 | 250 | 1252 | 2 |
| mread-2 | 44 | 36 | 18856 | 2786 | 8 | 1052 | > 9 h | > 24 h | 34 |
| mread-3 | 43 | 36 | 21848 | 2802 | 10 | 1102 | > 9 h | > 24 h | 35 |
| par4 | 31 | 28 | 1303 | 95 | 4 | 500 | 80 | 271 | 1 |
| par8 | 66 | 52 | $1.7\,10^{6}$ | 342 | 8 | 920 | - | - | 7 |
| par16 | 130 | 100 | $2.8\,10^{12}$ | 1190 | 16 | 1768 | - | - | 45 |

### D. Efficiency of the Cube Approximations

We have analyzed the efficiency of approximating the binary codes of a reachability graph by sets of cubes. This is achieved by comparing the number of required cubes versus the number of nodes in the STG and the number of reachable markings versus the number cubes. The cube comparison is done separately for two classes of STG's, those with

TABLE VII
CPU TIME FOR SYNTHESIS: SCALABLE EXAMPLES (∗NON-FC STG's)

| STG | P | T | RG | C | SM | CPU |
|---|---|---|---|---|---|---|
| phil3(*) | 39 | 30 | 864 | 123 | 9 | 2 |
| phil10(*) | 130 | 100 | $7.4\,10^{9}$ | 460 | 30 | 120 |
| phil20(*) | 260 | 200 | $5.5\,10^{19}$ | 820 | 60 | 1455 |
| muller10 | 40 | 20 | 420 | 100 | 16 | 1 |
| muller50 | 200 | 100 | $1.7\,10^{11}$ | 500 | 96 | 85 |
| muller100 | 400 | 200 | $1.2\,10^{27}$ | 1000 | 196 | 1437 |

TABLE VIII
TRADEOFFS AMONG MARKINGS, NODES, AND CUBES

| RG | cubes/node | markings/cube | markings/node |
|---|---|---|---|
| $\geq 10^6$ | 2.6 | $4\,10^{11}$ | $1.5\,10^{11}$ |
| $< 10^6$ | 2.4 | 1.7 | 0.7 |

less than $10^6$ markings and those surpassing this limit (see Table VIII).

For small benchmarks, we have reached a *cubes/node* ratio closer to 2.4, while the *markings/cube* ratio is closer to 1.7. Therefore, we can conclude that for small STG's, there are no significant differences between using the reachability graph or the proposed structural techniques. On the other hand, for larger benchmarks, the *cubes/node* ratio is closer to 2.6, while the *markings/cube* ratio is closer to $4 \times 10^{11}$. Thus, each node requires 2.6 cubes, and each cube approximates up to $4 \times 10^{11}$ markings—therefore justifying the efficiency of the cover-approximations methodology.

## X. CONCLUSIONS

Structural techniques for the analysis and synthesis of STG's are essential when the size of the state space becomes unmanageable. The proposed structural techniques intend to fill the gap between the STG's that can be analyzed by current state-based techniques and the existing STG's specifications of complex systems.

This work has presented new methods to synthesize STG's whose underlying PN is free choice. The proposed algorithms have polynomial complexity in the size of the net and can be easily extended to the class of PN's that can be covered by SM-components, although the existence of a SM-cover cannot be guaranteed for any nonfree-choice Petri net.

The experimental results show that the proposed methods obtain area-efficient implementations in short CPU times. Most of the existing tools were unable to synthesize the largest circuits, whereas the presented method is able to do it in few seconds. Future work will be devoted to fully characterize the class of Petri nets that can be handled by the presented techniques.

## APPENDIX
## MINIMIZATION TECHNIQUES

This Appendix will provide an overview of the minimization techniques that are structurally applied to simplify the covers used in an *atomic complex gate per excitation region* architecture.

### A. Basic Concepts

To efficiently implement this architecture, output signal transitions are partitioned into sets of *transition clusters* [34], [35]. Each cluster implies a complex gate for its implementation. These complex gates are combined by OR to form the set and reset functions, respectively.

*Definition 17 (Transition Cluster):* We define the transition clusters as a total partition $\{T^1_{a+}, \cdots, T^n_{a+}, T^1_{a-}, \cdots, T^m_{a-}\}$ of the rising and falling transitions of one output signal $a$, which must satisfy the following conditions.

1) Every rising or falling cluster contains at least one transition.
2) Every rising (falling) transition must be in one and only one transition cluster strictly composed of other rising (falling) transitions of the same signal.

A transition cluster, whether or not it contains rising or falling transitions, will be simply denoted by $T^i_{a^*}$. Superscripts are used to differentiate clusters of the same signal. All signal region definitions (ER's, QR's, QR$^r$, etc.) and implementability conditions can be easily extended to the usage on transition clusters.

Fig. 4(b) and (c) shows two different implementations for output signal $d$ in Fig. 1. The first implementation [Fig. 4(b)] corresponds to the transition cluster partitioning $T^1_d+ = \{d+/1\}$, $T^2_d+ = \{d+/2\}$, and $T^1_d- = \{d-\}$, which are implemented by covers $C(T^1_d+) = a + \bar{b}c$, $C(T^2_d+) = a$, and $C(T^1_d-) = \overline{a+b+c}$. This circuit is not SI because if the AND-OR gate for $T^1_d+$ is slow enough, the pulse on input $a$ can propagate to the output $S_d$. In Fig. 4(c), transitions $d+/1$ and $d+/2$ are merged into one cluster $T_d+ = \{d+/1, d+/2\}$. This makes the overall circuit simpler and SI (the races between inputs $a$ and $b$ take place only within one AND–OR gate).

### B. Complete Region Covers

Generating complete covers for all the rising or falling transitions of an output signal is one of the efficient minimization techniques that can be applied. In that case, the circuit can be exclusively created by using the corresponding set or reset function [5]. Every cover is checked to be *complete* by analyzing that all markings in QR($T^i_{a^*}$) are covered by $C(T^i_{a^*})$.

If all rising covers are complete, the set function implements the circuit. Similarly, if all falling covers are complete, the reset function can be alternatively used. In case both rising and falling functions are complete, the smallest or faster function should be selected.

### C. Region Expansions

Circuits can be minimized by *expanding* the region covers toward the quiescent region and the dc-*set*. All transformations are characterized by either the elimination of a signal from the support of the function or the elimination of literals from the cubes. The main objective of expanding is to simplify the covers but also to obtain *complete* region covers with the subsequent minimization (allows a combinational implementa-

tion for the signal). Therefore, this minimization has a higher priority than other transformations.

Transition clusters are *merged* together when the complexity of the resulting region cover decreases. Transition clusters $T^i_{a^*}$ and $T^j_{a^*}$ are merged, creating a new cluster $T^k_{a^*} = T^i_{a^*} \cup T^j_{a^*}$ with covers $C(T^k_{a^*}) = C(T^i_{a^*}*) + C(T^j_{a^*})$, and eliminating the seminal ones. Merging requires checking whenever the resulting cover can be positively matched in the gate library. Merging also allows one to derive an increased number of complete covers.

### D. Collapsing of Memory Elements

The structure of the architecture and the behavior of the C-latch can be used to further simplify the circuit [24]. Consider the signal network for an output signal $a$ implemented by a C-latch with equation $a = S(a)\overline{R(a)} + a(S(a) + \overline{R(a)})$, and set and reset networks with one region cover each $S(a) = v_1v_2$, $R(a) = \overline{v}_1\overline{v}_2$. Both cubes can be collapsed into the C-latch: $a = v_1v_2 \cdot \overline{\overline{v}_1\overline{v}_2} + a(v_1v_2 + \overline{\overline{v}_1\overline{v}_2})$, obtaining $a = v_1v_2 + a(v_1 + v_2)$. Hence, both set and reset region networks can be substituted by $S(a) = v_1$, $R(a) = v_2$, being $S(a)$ and $R(a)$.

Similarly, if the set and reset networks $S(a) = v_1v_2$, $R(a) = v_1\overline{v}_2$, their cubes have the same support and are at distance one. Again, both cubes can be collapsed into the C-latch, obtaining $a = v_1v_2 + \overline{v}_1a$. Then, both set and reset region networks can be substituted by the expressions $D(a) = v_2$, $G(a) = v_1$, being $D(a)$ and $G(a)$ the data and control inputs of a gated latch that replaces the initial C-latch.

### E. Backward Region Expansions

The *backward quiescent region* $\mathrm{BR}(a_{i^*})$ of a transition is the maximal connected set of markings that can reach $\mathrm{ER}(a_{i^*})$ without enabling any other transition $a_{j^*}$.

Further circuit minimizations can be obtained if a cover $C(T^i_{a^*})$ is also extended to cover markings in its backward quiescent region $\mathrm{BR}(T^i_{a^*})$. Covering markings in the backward quiescent regions is only possible because of the characteristics of the C-latch (as pointed out by [19] and [36]). Maintaining one of the inputs of the C-latch at 1 (0) while its output is at 1 (0) creates and extra *observability* dc-*set* that can be used to further minimize circuits.

Given the architecture in Fig. 3(c), if both the set cover and the output are still at 1, activating the reset cover will not produce a falling transition of the output until the set cover falls to 0. Hence, the cover of any falling transition can be activated before reaching its ER, but only if it can be guaranteed that the set cover will remain at 1 until the falling transition is excited. Similar conditions apply for rising transitions.

Similar *monotonicity* conditions are required in the backward regions; that is, the cover changes exactly twice in any sequence, where the rising change is at a marking in $\mathrm{BR}(T^i_{a^*}) \cup \mathrm{ER}(T^i_{a^*})$ and the falling change in $\mathrm{QR}(T^i_{a^*})$.

The *backward quiescent region* $\mathrm{BR}(a_{i^*})$ can be structurally defined by the *backward quiescent place set* $\mathrm{BPS}(a_{i^*})$. A place belongs to $\mathrm{BPS}(a_{i^*})$ if it is interleaved between $a_{j^*}$ and $a_{i^*} \in \mathrm{next}(a_{j^*})$, i.e., $\mathrm{BPS}(a_{i^*}) = \{p \mid \exists a_{j^*}, a_{i^*} \in \mathrm{next}(a_{j^*})\colon p \in \mathcal{IR}(a_{j^*}, a_{i^*})\}$. The same concept can be extended to transition clusters and to restricted regions, i.e., $\mathrm{BPS}(T^i_{a^*}) = \bigcup_{a_{j^*} \in T^i_{a^*}} \mathrm{BPS}(a_{j^*})$.

Last, it is also essential to determine which are the markings that the predecessor transition clusters are covering to determine the subset of the BR region that is allowed to be covered. For each place $p$ in $\mathrm{BPS}(a_{i^*})$, we will define by $cv^b(p)$ the subset of markings in $\mathrm{MR}(p)$ covered by some predecessor transition $(a_{j^*} \in prev(a_{i^*}))$: $\forall a_{j^*} \in prev(a_{i^*})$: $[\forall p \in \mathrm{QPS}(a_{j^*}) \cap \mathrm{BPS}(a_{i^*})\colon cv^b(p) = cv(p) \cdot C(a_{j^*})]$. Once we have computed these subsets, the correct covering of markings in the backward quiescent region is straightforward: $\forall p \in \mathrm{BPS}(a_{i^*})\colon cv(p) \cdot C(a_{i^*}) \subseteq cv^b(p)$.

### F. Technology Mapping

Circuits generated after the overall minimization process are mapped onto the technology provided by the designer. Blocks in the signal network can be combined in single cells when available in the library of existing gates. This cell-binding process provides an extra degree of minimization by substituting several logic blocks in the signal network by a more efficiently implemented cell in the library. A technology mapper tailored for SI-circuits has been developed following the Boolean matching techniques proposed in [33]. However, note that it is not possible to apply a generalized decomposition process of the blocks in the signal network due to the restrictive correctness conditions imposed by speed-independent circuits [37].

## REFERENCES


[1] A. J. Martin, "Formal program transformations for VLSI circuit synthesis," in *Formal Development of Programs and Proofs*, E. W. Dijkstra, Ed. Reading, MA: Addison-Wesley, 1989, pp. 59–80.

[2] C. A. Petri, "Kommunikation mit Automaten," Ph.D. dissertation, Institut für Instrumentelle Mathematik, Bonn, 1962, Tech. Rep. Schriften des IIM Nr. 3.

[3] M. A. Kishinevsky, A. Y. Kondratyev, and A. R. Taubin, "Formal method for self-timed design," in *Proc. Eur. Design Automation Conf.*, Feb. 1991, pp. 197–201.

[4] L. Y. Rosenblum and A. V. Yakovlev, "Signal graphs: From self-timed to timed ones," in *Proc. Int. Workshop Timed Petri Nets*, July 1985, pp. 199–206.

[5] T.-A. Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, June 1987.

[6] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuits synthesis," University of California, Berkeley/ERL, Tech. Rep. M92/41, May 1992.

[7] P. A. Beerel and T. H. Meng, "Automatic gate-level synthesis of speed-independent circuits," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, IEEE Computer Society Press, Nov. 1992, pp. 581–586.

[8] C. Ykman-Couvreur, B. Lin, and H. De Man, "ASSASSIN: A synthesis system for asynchronous control circuits," IMEC, Sept. 1994, Tech. Rep., user and tutorial manual.

[9] K.-J. Lin and C.-S. Lin, "Automatic synthesis of asynchronous circuits," in *Proc. ACM/IEEE Design Automation Conf.*, IEEE Computer Society Press, June 1991, pp. 296–301.

[10] C. Ykman-Couvreur, B. Lin, G. Goossens, and H. De Man, "Synthesis and optimization of asynchronous controllers based on extended lock graph theory," in *Proc. Eur. Conf. Design Automation (EDAC)*, Feb. 1993, pp. 512–517.

[11] M. Hack, "Analysis of production schemata by Petri nets," M.S. thesis, Massachusetts Institute of Technology, Cambridge, Feb. 1972.

[12] J. Desel and J. Esparza, *Free Choice Petri Nets.* Cambridge, U.K.: Cambridge Univ. Press, 1995.
[13] F. García-Vallés and J. M. Colom, "A Boolean approach to the state machine decomposition of Petri nets with OBDD's," in *Proc. 1995 IEEE Int. Conf. Systems, Man and Cybernetics*, Oct. 1995.
[14] P. Vanbekbergen, "Optimized synthesis of asynchronous control circuits from graph-theoretic specification," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, Nov. 1990, pp. 184–187.
[15] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Norwell, MA: Kluwer Academic, 1984.
[16] F. M. Brown, *Boolean Reasoning: The Logic of Boolean Equations.* Norwell, MA: Kluwer Academic, 1990.
[17] V. I. Varshavsky, *Self-Timed Control of Concurrent Processes.* Norwell, MA: Kluwer Academic, 1990.
[18] K. Lautenbach, "Linear algebraic techniques for place/transition nets," in *Petri Nets: Central Models and their Properties, Advances in Petri Nets 1986*, W. Brauer, W. Reisig, and G. Rozenberg, Eds., vol. 254 of *Lecture Notes in Computer Science.* Berlin, Germany: Springer Verlag, 1987, pp. 142–167.
[19] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev, "Basic gate implementation of speed-independent circuits," in *Proc. ACM/IEEE Design Automation Conf.*, June 1994, pp. 56–62.
[20] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, "Concurrent hardware. The theory and practice of self-timed design," *Series in Parallel Computing.* New York: Wiley, 1994.
[21] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits.* Norwell, MA: Kluwer Academic, 1993.
[22] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A unified signal transition graph model for asynchronous control circuit synthesis," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, IEEE Computer Society Press, Nov. 1992, pp. 104–111.
[23] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1185–1205, Nov. 1989.
[24] P. A. Beerel, "CAD tools for the synthesis, verification, and testability of robust asynchronous circuits," Ph.D. dissertation, Stanford University, Stanford, CA, Aug. 1994.
[25] S. Burns, "General conditions for the decomposition of state holding elements," in *Proc. Int. Symp. Advanced Research in Asynchronous Circuits and Systems*, Aizu, Japan, Mar. 1996, pp. 48–57.
[26] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, E. Pastor, and A. Yakovlev, "Decomposition and technology mapping of speed-independent circuits using Boolean relations," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, Nov. 1997, pp. 220–227.
[27] E. Pastor and J. Cortadella, "Polynomial algorithms for the synthesis of hazard-free circuits from signal transition graphs," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, Santa Clara, USA, IEEE Computer Society Press, Nov. 1993, pp. 250–254.
[28] E. Pastor, J. Cortadella, A. Kondratyev, and O. Roig, "Structural methods for the synthesis of speed-independent circuits," in *Proc. Eur. Design Test Conf. (EDAC-ETC.-EuroASIC)*, Paris, France, Mar. 1996, pp. 340–347.
[29] A. Kovalyov and J. Esparza, "A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs," in *Proc. Int. Workshop Discrete Event Systems, WODES'96*, Aug. 1996, pp. 1–6.
[30] E. Pastor and J. Cortadella, "An efficient unique state coding algorithm for signal transition graphs," in *Proc. IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1993, pp. 174–177.
[31] J. Esparza and M. Silva, "A polynomial-time algorithm to decide liveness of bounded free choice nets," *Theoretical Comput. Sci.*, no. 102, pp. 185–205, Apr. 1992.
[32] O. Roig, J. Cortadella, and E. Pastor, "Verification of asynchronous circuits by BDD-based model checking of Petri nets," in *Proc. 16th Int. Conf. Application and Theory of Petri Nets*, Torino, June 1995, vol. 935 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 374–391.
[33] F. Mailhot and G. De Micheli, "Technology mapping using Boolean matching," in *Proc. Eur. Conf. Design Automation (EDAC)*, Glasgow, U.K., Mar. 1990, pp. 180–185.
[34] E. Pastor, J. Cortadella, and O. Roig, "A new look at the conditions for the synthesis of speed-independent circuits," in *Proc. 5th Great Lakes Symp. VLSI*, Buffalo, NY, May 1995, pp. 230–235.
[35] A. Kondratyev, M. Kishinevsky, and A. Yakovlev, "On hazard-free implementation of speed-independent circuits," in *Proc. ASP-DAC'95*, Aug. 1995, pp. 241–248.
[36] P. A. Beerel and T. H.-Y. Meng, "Logic transformations and observability don't cares in speed-independent circuits," in *ACM Int. Workshop Timing Issues in the Specification and Synthesis of Digital Systems*, Sept. 1993.
[37] P. Siegel and G. De Micheli, "Decomposition methods for library binding of speed-independent asynchronous designs," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1994.

**Enric Pastor** received the M.S. and Ph.D. degrees in computer science from the Universitat Politécnica de Catalunya, Barcelona, Spain, in 1991 and 1996, respectively.

He is an Associate Professor in the Department of Computer Architecture of the Universitat Politécnica de Catalunya. He was a Visiting Scholar at the University of Colorado at Boulder, CO, and the Inter-university Microelectronics Centre (IMEC), Belgium, in 1992 and 1994, respectively. In 1988, he was a Leverhulme Trust Fellow visiting the University of Newcastle upon Tyne, U.K. His research interests include formal methods for the computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits and concurrent systems.

**Jordi Cortadella** (S'87–M'88) received the M.S. and Ph.D. degrees in computer science from the Universitat Politécnica de Catalunya, Barcelona, Spain, in 1985 and 1987, respectively.

He is an Associate Professor in the Department of Software of the Universitat Politécnica de Catalunya. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include computer-aided design of VLSI systems with special emphasis on synthesis and verification of asynchronous circuits, concurrent systems, computer arithmetic, and parallel architectures. He has coauthored more than 80 research papers in technical journals and conferences. He has served on the technical committees of several international conferences in the field of design automation and concurrent systems.

**Alex Kondratyev** (M'97), for a photograph and biography, see p. 771 of the September 1998 issue of this TRANSACTIONS.

**Oriol Roig** received the engineer in computer science degree in 1991 and the Ph.D. degree in computer science in 1997, both from the Universitat Politécnica de Catalunya, Barcelona, Spain.

He was an Assistant Professor at the Universitat Politécnica de Catalunya until May 1998, when he joined the Methodology group at National Semiconductor, Santa Clara, CA. His research interests include asynchronous and formal hardware verification.